«Инфра-Инженерия»

А. В. Протодьяконов
П. А. Пылов
В. Е. Садовников

АЛГОРИТМЫ DATA SCIENCE

И ИХ ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ НА PYTHON

А. В. Протодьяконов
П. А. Пылов
В. Е. Садовников

АЛГОРИТМЫ DATA SCIENCE

И ИХ ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ НА PYTHON

Учебное пособие

Москва Вологда
«Инфра-Инженерия»
2022

УДК 004.89
ББК 32.813
П83

Рецензенты:

кандидат технических наук, доцент, заведующий кафедрой
информационных и автоматизированных производственных систем
ФГБОУ ВО «Кузбасский государственный технический университет»
И. В. Чичерин;

доктор технических наук, профессор РАН, главный научный сотрудник
ФГБНУ «Федеральный исследовательский центр угля и углехимии
Сибирского отделения Российской академии наук»
А. Е. Майоров

Протодьяконов, А. В.

П83 Алгоритмы Data Science и их практическая реализация на Python : учебное пособие / А. В. Протодьяконов, П. А. Пылов, В. Е. Садовников. – Москва ; Вологда : Инфра-Инженерия, 2022. – 392 с. : ил., табл.

ISBN 978-5-9729-1006-9

Рассмотрен полный каскад разработки моделей искусственного интеллекта. Проанализирована область Data Science, из которой выделены все необходимые для прикладной сферы алгоритмы машинного обучения, расположенные по уровню возрастания сложности работы с ними.

Для студентов, изучающих информационные технологии. Может быть полезно как начинающим программистам, так и специалистам высокого уровня.

УДК 004.89
ББК 32.813

ISBN 978-5-9729-1006-9

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Данное учебное пособие предполагает обучение студентов технических специальностей (и является профильным для студентов информационных технологий) навыкам и методикам работы с сложными моделями машинного и глубокого обучения.

Отличительной чертой пособия от других схожих курсов обучения искусственному интеллекту (онлайн площадки, интернет-курсы) является плавность возрастания сложности и целостность логики повествования. Сложность входного порога к изучению алгоритмов и моделей машинного / глубокого обучения является минимальной из всех возможных, то есть даже неподготовленный студент может начать обучению по настоящему учебному пособию и плавно упрочнять свои навыки до уровня Junior Data Scientists.

Состав и последовательность вариантов работ в учебном пособии организованы по логическому возрастанию уровня сложности. Таким образом, предложенные задания могут быть освоены студентами не только в рамках курса дисциплины, но и, по своему интересу, студенты могут самостоятельно повышать свои навыки и умения на более сложных заданиях, не затрачивая дополнительное время на поиск заданий и наборов данных в сети Интернет.

Полное освоение работ по данному учебному пособию (при условии самостоятельного выполнения практических работ) гарантирует появление у студента навыков и умений hard skills, необходимых и достаточных для трудоустройства в компании по специальности Junior Data Scientists / Data Analyst[1].

[1] Согласно реальным данным анкет одного из крупнейших в РФ поисковых сервисов работы (https://spb.hh.ru/vacancies/data-analyst и https://spb.hh.ru/search/vacancy?clusters=true&area=2&ored_clusters=true&enable_snippets=true&salary=&text=data+scientists) по состоянию на конец ноября – начало декабря 2021 года.

Часть 1
ПРОЦЕСС МАШИННОГО ОБУЧЕНИЯ

ЗАДАЧИ МАШИННОГО ОБУЧЕНИЯ

Начинать знакомство с машинным обучением логично с обзора тех задач (классификации задач), которые относятся к машинному обучению. Данным задачам и методам их решения будут посвящены следующие разделы учебного пособия.

Классификаций задач в машинном обучении очень много (некоторые идут от способа решения, другие от метода решения), в нашем случае примем за основу самую общую и наиболее простую классификацию, выделяющую три основных группы в задачах машинного обучения:

1. Обучение с учителем.
2. Обучение без учителя.
3. Обучение с подкреплением.

Начнем с первой группы задач. Такая концепция подразумевает под собой тот факт, что у нас есть на входе некоторые данные, для которых известен ответ (задача очень похожа на арифметическую / геометрическую прогрессию, когда нужно угадать следующее/предыдущее число в ряду чисел). При этому у нас есть некоторые новые данные, для которых требуется найти правильное значение (наиболее правильное), так как машинное обучение предполагает возможность, при котором может не быть однозначно правильного ответа на поставленный вопрос. То есть существует некоторый набор ответов, каждый из которых по-своему близок к правильному, но, скорее всего, правильным не является (всегда существует определенная статистическая ошибка).

Обучение с учителем – наиболее простой класс задач в машинном обучении, так как предполагает, что у нас уже есть собранные данные, на эти данные есть правильный ответы и по аналогичным данным нам нужно предсказать ответ, который будет похож на правильный и максимально близок к правильному (во всяком случае мы сможем оценить, насколько предсказываемый

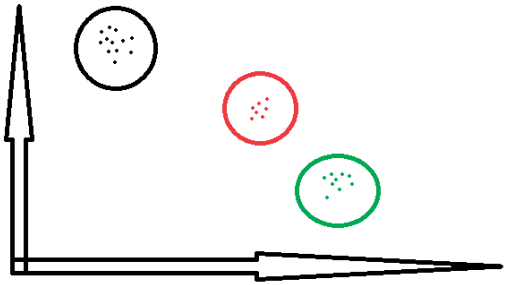
ответ близок к правильному). В обучении с учителем выделяют две основные задачи:

- задача регрессии;
- задача классификации.

По сути, они достаточно похожи друг на друга, только в задачи регрессии мы предсказываем числовое значение, а в задачи классификации ответ представлен в виде ранга или категории (по факту – нечисловое значение). Задачи решаются одними и теми же алгоритмами, но принято их разделять и говорить, что если предсказывается числовое значение – то это точно задача регрессии (у неё свой набор методов – линейная регрессия, нелинейная регрессия), в то время как предсказание рангов и классов относится к задаче классификации (у неё существует другой набор алгоритмов – метод ближайших соседей, множественная классификация).

Обучение без учителя – вторая группа задач – говорит, что у нас нет правильного ответа. Существуют данные, но ответа на них нет. Соответственно возникает ряд задач в связи с тем, что у нас нет ответа на данные.

Первая подзадача – задача кластеризации. В ней по определённым параметрам (например, количество кластеров) необходимо предсказать каким образом данные распределены между собой. Это позволяет разнести данные по кластерам, основываясь на среднем удалении от центра кластера, на количестве кластеров и других признаков, тем самым решив задачу кластеризации таким образом.



Кроме того, группа задач обучения без учителя включает в себя методы понижения размерности, когда существует огромное количество признаков и нам требуется уменьшить их до нескольких, чтобы потом применить задачу кластеризации или задачу обучения с учителем (чтобы на разных параметрах определить какие параметры важны и, снизив размерность, решить задачу обучения с учителем. При большой размерности решить задачу с учителем не представляется возможным без серьезного аппаратного обеспечения).

Также к классу задач обучения без учителя относится класс задач по выявлении аномалий. Они похожи на кластеризацию, где есть все данные и дополнительно могут быть ещё какие-то. Обращаем особое внимание на то, что

«каких-то дополнительных данных» может и не быть, то есть мы можем не знать какая конкретно будет аномалия, однако мы можем предположить, что некоторые зависимости в данных уже являются аномалиями. Например, задача Фродо относится к классу задач обучения без учителя, так как в ней нужно предсказать пытается ли кто-то обмануть систему или нет. Из практики можно привести пример автоматического распознавания Ddos-атаки, когда алгоритм сам определяет, что такого количества посетителей не должно быть и, скорее всего прирост запросов на сайт – действие злоумышленников.

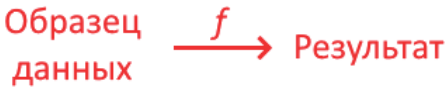
Третий тип задач машинного обучения – это обучение с подкреплением. Оно похоже на обучение, но это его единственное сходство с двумя другими категориями, потому что обучение с подкреплением предполагает, что у нас нет данных как таковых, но у нас есть среда, которая генерирует поток сигналов. На основании данных сигналов система должна принимать решения самостоятельно.

Разумеется, задачу третьего типа можно свести к одной из двух типов задач машинного обучения, однако важное отличие третьего типа от двух остальных состоит в том, что существует положительная обратная связь от среды. Обучение предполагает, что у нас будет ответ, насколько мы правильно или неправильно решили поставленную от среды задачу (правильно повернули на автомобиле или нет). Подкрепление есть как система обратной связи – положительной или отрицательной; в любом случае у нас должен существовать новый ассортимент методов и алгоритмов для решения узконаправленной задачи.

МОДЕЛЬ И ПРОЦЕСС МАШИННОГО ОБУЧЕНИЯ

Модель машинного обучения – это некоторая функция или алгоритм (порядок действий), который по набору входных параметров (десятки, сотни, тысячи значений) выдает какой-то ответ или оценку ситуации (какое-то число; действия, которые необходимо предпринять; категорию; набор ответов).

Функция ставит в соответствие некоторый образец данных некоторому решению (решение может быть достаточно произвольным: число, действие, категория, отсутствие действия).



Этому результату модель может обучаться (может не обучаться, а просто выдавать его).

Чтобы получить эту главную функцию необходимо выполнить определенный, достаточно трудоёмкий процесс.

Процесс всегда начинается с выбора источника данных, а также обязательная постановка цели, метрики или критерия. То есть того, каким должен быть результат, который нам нужен (насколько точный он должен быть; путь, который он должен проходить). В любом случае всегда должна стоять некоторая цель или метрика результата. Каким образом эта цель (метрика) формулируется в конкретных задачах машинного обучения будет исследовано в следующих главах учебного пособия, но очень важно, что перед тем, как начинать делать модель машинного обучения, необходимо четко понимать, что от данной модели нам необходимо получить.

Без двух вышеописанных основополагающих блоков (цель и выбор источника данных) процесс машинного обучения теряет смысл.

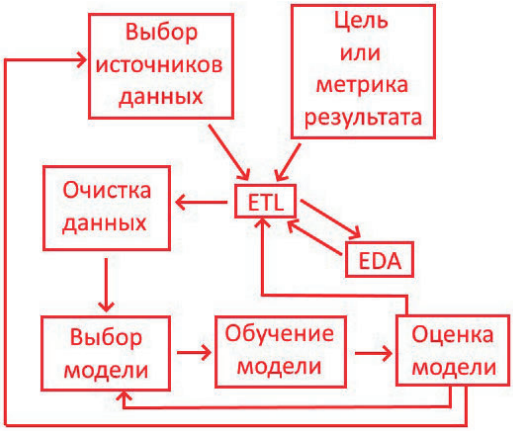
После того, как определены два блока, наступает процесс сбора, очистки и объединения данных (процесс ETL, подробно рассмотрен в следующих главах); после него может идти процесс исследовательского анализа данных (EDA) – в случаях, если данных очень много и необходимо предварительно найти взаимосвязи в данных, исключить лишние информационные ветви. Также процесс EDA может вернуть какие-то данные вновь в процесс обработки.

То есть, поменяв что-то в процессе обработки данных, вернули результат и далее, основываясь на результате двигаемся дальше.

Затем следует процесс очистки данных (допустим, перевод категориальных данных в числовые).

После очистки данных можно произвести выбор модели машинного обучения и произвести её обучение. После обучения обязательно следует проверка результата по заранее выставленной метрике (для каждого алгоритма существует своя собственная метрика. Например, для задачи регрессии метрикой может выступать метод наименьших квадратов; для задачи классификации – матрица неточности).

Завершающий этап – оценка модели. После оценки модели можно вновь вернуться к выбору модели, если оценка является неудовлетворительной или улучшить данные на разных этапах (в следующих главах подробнее об этом).



После всех этапов, в случае, когда нас устраивает оценка модели, модель может быть внедрена в виде черного ящика/набора функций/оборудования, на котором работает нейросеть отгружается на операционное устройство для выполнения интеграции с сервисом (мобильная платформа, ПК и так далее).

ПОНЯТИЕ ETL
ETL (Extract, Transformer, Load)

Процесс ETL – Extract, Transform, Load. В переводе с английского оригинала звучит как «Выбрать, Преобразовать, Загрузить» – процесс препреподготовки данных, когда нам необходимо понять всё ли у нас готово для того, чтобы начинать работать с моделью.

Процесс ETL обычно включает некоторый первичный анализ исходных данных. Допустим, у нас существуют данные по энергопотреблению зданий, но в них нет погодных данных в выборке. У нас есть гипотеза, что наличие погодных данных так или иначе позволит уточнить модель. В таком случае, на этапе процесса ETL необходимо добавить погодные и гидрометеорологические данные, чтобы ими обогатить модель и проверить больше гипотез на этапе построения модели.

К этапу сбору и подготовки данных допустимо возвращаться по несколько раз по ходу оценки точности работы модели.

Однако, процесс является базовым и состоит из нескольких инструментов. Первый из них – это получить данные из нескольких источников. То есть настройка получения данных, так как процесс может выполняться множество раз, нам требуется процедура, позволяющая регулярно получать данные.

Вторым аспектом является очистка данных, либо заполнить пропуски при существовании таких. Пропуски являются наиболее частой проблемой

при работе с моделью машинного обучения. Существует несколько вариантов заполнения пропусков:

1. Провести интерполяцию значений (заполнение близкими значениями). Предположить, что у нас также развивалась ситуация и заполнить наиболее характерное значение. Не работает для категориальных данных! Однако хорошо справляется с числовыми или ранговыми значениями.
2. Заполнить пропуски никогда не встречающимися значениями (−1 или −100). Таким образом мы отмечаем, что эти данные никак не должны участвовать в оценке модели, но при этом они не должны испортить сам модель – сам пропуск данных «выключает» тот экземпляр данных (или целый кортеж), который есть с этим пропуском.

При преобразовании данных всегда следует помнить о том, что данные нельзя «ломать». Намеренная замена пропусков отличными от реальности значениями приведет к основному правилу машинного обучения «мусор на входе – мусор на выходе», что отличным образом описывает качество модели машинного обучения. Поэтому данные должны остаться корректными, сохраняя в себе свою главную суть, а не просто поправленными по нашей необходимости.

Третьим аспектом в ETL является объединение данных из нескольких источников. Предположим, существуют данные по городу, людям и автомобилям. Чтобы провести аналитику замкнутого цикла человек – автомобиль – город, достаточно будет объединить данные по человеку, автомобилю, который с ним связан и городу, в котором он проживает.

Таким образом, произойдет расширение параметров, но модель сможет работать с каждым кортежем данных атомарно, что позволит распараллелить процесс и разбить данные на тренировочную и тестовую выборку без потерь.

На этапе ETL, при необходимости, должно быть объединение данных, чтобы каждая строка данных исчерпывающим образом описывала ситуацию предметной области без пропусков в них.

Только после подготовки данных можно переходить к непосредственному построению модели машинного обучения.

ПОНЯТИЕ EDA
Что такое EDA (Exploratory Data Analysis)

Важной частью процесса машинного обучения является разведка или исследовательский анализ данных (EDA, от английского Exploratory Data Analysis – Исследовательский анализ данных).

Зачем он нужен? Он позволяет на самом деле заглянуть немножко вперед, как бы представить, что вдруг у нас модель машинного обучения уже построена, что она из себя могла бы представлять.

Иногда на этапе разведывательного анализа данных у нас получается выбрать некоторую очень простую модель, то есть увидеть какую-то простую взаимосвязь в исходных данных, которая позволит получить точный ответ. Это бывает редко, но это одна из задач исследовательского анализа данных.

Вторая задача, на самом деле более важная задача, — это найти пропуски в данных. Поскольку с пропусками мы не можем построить адекватную модель, то нам нужно провести анализ данных и найти все пропуски во всех сериях, во всех параметрах, которые у нас присутствуют и, возможно, это будет даже не явные пропуски, но какие-то нулевые значения, например, или любые, скажем так, некорректные значения, которые нам нужно на этапе очистки данных каким-то образом преобразовать.

Исследовательский анализ данных проводится обычно единожды для каждой конкретной задачи машинного обучения на основании этих данных что у нас есть и его результаты (они представлены в виде некоторого отчета) обычно появляется ряд графиков и замечаний и по его итогам идут исправления, либо улучшения.

То есть EDA сам по себе не строит никакую модель, однако, он позволяет существенным образом улучшить данные, которые для этой модели делаются:

1. Первое улучшение: находим пропуски в данных.
2. Второе: оценивается параметр распределения наших параметров, то есть какие это распределения (равномерное и т. д.).
3. Корреляции. Причём рассматриваются как корреляции между самими параметрами (чтобы понять какие из них, например, можно отбросить), так и на корреляцию между предсказываемыми значениями (в случае обучения с учителем, и даже обучение с подкреплением. Предсказываемое значение одного из параметров, чтобы понять, какую самую простую модель мы можем использовать по параметрам поиска по неизвестным причинам, которые в наибольшей степени коррелирует с результатом).

Собственно, это одна из задач исследовательского анализа данных: помочь определиться с наиболее важными параметрами, в том числе за счет корреляции между этими параметрами и предсказываемым значением.

И, естественно, здесь (в EDA) смотрят на примеры данных, потому что иногда эти три ранее представленные метода не позволяют что-то выявить. Однако на примерах данных, то есть на нескольких образцах данных легко видеть одни и те же значения параметров и какую-то сложную корреляционную связь между рядом параметров.

Поэтому всегда исследовательский анализ данных включают примеры того, что за данные будут использоваться, как они выглядят, могут ли быть какие-то ограничения на них.

Например, если у нас есть задача сегментации изображения, то нам важно оценить какая в задаче область сегмента. И здесь только по примерам данных мы сможем понять, на фотографии какая максимальная область сег-

ментации, поэтому уже по оценке этой области можно на исходные данные наложить какие-то ограничения. Без примеров данных это сделать очень сложно или даже невозможно (основываясь только на пропусках параметров или на корреляции).

В любом случае, следует помнить, что разведывательный анализ данных выполняется единожды и его результаты используются в дальнейшем при построении всей модели машинного обучения.

ПОДГОТОВКА ДАННЫХ

Предобработка (препроцессинг) данных является логическим продолжением процесса ETL и в большинстве случаев его даже объединяют с этим процессом, однако, принято разделять эти два понятия, потому что в процессе ETL нам нужно получить чистые данные (заполнить пропуски, найти опечатки, объединить все данные в источники), на этапе подготовки данных мы эти данные уже соответствующим образом обрабатываем, преобразовываем, переформатируем, чтобы их можно было подать на вход модели машинного обучения. И именно это форматирование, что существенно, сильно зависит от используемой модели машинного обучения.

Например, ряд моделей не требует разбиения категориальных данных на One Hot Vector, они могут работать с категориальными данными непосредственно так, как вы их передаете.

Другие модели, например, линейной регрессии не умеют работать с категориальными данными, поэтому данные нужно предварительно обработать.

Также, в ряде случаев, у нас данные могут быть уже примерно одного диапазона – нам их не нужно будет формировать.

В другом случае нам переменные в исходных данных не помогают уточнить постановку задач и не помогают повысить точность модели машинного обучения. Соответственно, этап обработки данных не всегда является обязательным, однако, в большинстве задач он требуется и обычно включают этот блок обработки данных. Всё вместе, в большинстве решений задач машинного обучения, называют препроцессингом данных.

На практике, это еще один шаг, который необходимо выполнить с данными, но он не всегда требуется.

Итак, в основном выделяют три основных аспекта подготовки данных.

1. Нормирование данных, когда нам нужно их привести к одному диапазон. Математика очень любит работать с диапазоном от 0 до 1, соответственно, все пакеты они заточены на то, чтобы приводить данные к диапазону [0; 1].

Рассмотрим конкретный пример:

$$x : 1, 2, 4, 5$$
$$y : -100, 300, 1100$$

Нам нужно привести данные к одному диапазону. X у нас «маленький», а Y «большой», поэтому погрешность по Y будет превалировать над погрешностью по X. Требуется свести данные к диапазону от 0 до 1.

Для этого из всех значений вычитается минимальное, чтобы нам сдвинуть на ноль, а потом, соответственно, делится на разность между минимальным и максимальным значением.

$$x \to \frac{(x - \min)}{(\max - \min)}.$$

В первом случае (по X) получается следующее:

$$x : 1, 2, 4, 5 \to 0, \frac{1}{4}, \frac{3}{4}, 1.$$

Во втором случае (по Y):

$$y : -100, 300, 1100 \to 0, \frac{1}{3}, 1.$$

Как видно, цифры уже примерно одинаковые, поэтому с ними работать намного приятнее. Процесс называется нормированием, он подходит для всех числовых перемен.

Отметим, что если данные необходимо проследить после обучения модели, то необходимо провести процедуру обратного масштабирования. Так как нормализация относится к типу преобразования данных.

Если у нас данные распределены нормально и нам необходимо отбросить выбросы, то мы можем привести данные к диапазону [−3; 3] и затем просто отбросить крайние значения, чтобы у нас какие-то «хвосты», статистически выбросы, либо просто не характерные точки, которые нежелательны для модели, мы можем автоматически отбросить, если распределение близко к нормальному.

2. Категориальные данные. Это второй тип преобразования, который производится над данными. Категориальные данные приводят к единичным векторам (One Hot Vector).

Давайте посмотрим на конкретном примере, в чем это заключается. Предположим, что у нас есть 3 категории:

Категории	
пол	
1	Ж
2	М
3	Д

Категории распределены по полу (женский, мужской и другой). Пол – это категория. Не все модели машинного обучения умеют работают с категориальными данными. Чтобы нам эти данные подать всем моделям на вход, нам нужно преобразовать этот набор данных к другому набору данных (к так называемым единичным векторам).

Для этого мы также оставляем идентификатор и добавляем параметров по значениям категорий. По предыдущему примеру:

	пол_Ж	пол_М	пол_Д
1	1	0	0
2	0	1	0
3	0	0	1

Параметры по столбцам взаимоисключающие, поэтому в каждой строке будет только одна 1. Именно поэтому кодировка и называется One Hot Vector.

Преобразовав категориальные данные таким образом, мы их перевели из текстовых в числовые. Теперь, числовые мы можем подать на вход любой модели машинного обучения без каких-либо ограничений (будь то линейная регрессия, градиентный бустинг или что-то ещё).

3. К третьему типу преобразования относится **преобразование циклических переменных**. Циклическая переменная – это, например, роза ветров – направление ветра в градусах. То есть у нас направление ветра в градусах может меняться от 0° до 360°, при этом 360° = 0°.

Однако, проблема состоит в том, что, например, значения 355° и 5° находятся рядом на окружности, но на модели разрыв будет очень большой. Чтобы нам сообщить модели машинного обучения, эти данные – циклические, то для них применяют тригонометрические функции.

Предположим, что у нас параметр z характеризует градусы.

z	5°	120°	360°
cos z			
sin z			

Благодаря тригонометрическим функциям значения cos(355) и cos(5) будут лежать очень близко друг к другу по любой метрике.

Дополнительным преобразованием тригонометрических функций можно свести их значения к необходимому диапазону значений.

Такое преобразование позволяет корректно воспринимать данные моделями машинного обучения. Благодаря преобразованию тригонометрическими функциями модель сохраняет интерпретируемость результатов. Это очень важное свойство, поскольку, если данные будут плохо интерпретируемы (или неправильно интерпретируемы), то они только ухудшат модель.

Кроме градусов могут быть часы, минуты (время), дни в году (от 1 до 365), месяцы и так далее.

Запомните, что всё, что касается циклических переменных, должно быть преобразовано при помощи тригонометрических функций!

РАЗБИЕНИЕ ВЫБОРКИ

Ещё одним важным моментом в машинном обучении при работе с данными является разбиение данных на обучающие, проверочные и валидационные наборы.

Представим, что у нас есть *все* данные (миллион значений) и нам нужно по этим данным понять какая модель машинного обучения лучше работает и самое главное – какую точность эта модель дает. Если мы, например, обучаем модель на всем миллионе назначений, потом проверим как модель обучилась, то в ряде случаев можно получить точность в 100 %. И внешне всё кажется идеальным.

Но соотносится ли представленная точность с реальной жизнью будет совершенно непонятно. Модель может переобучиться (о переобучении в следующих главах) и её точность на всех данных, которые ей передали, будет максимальной, но при этом она не будет обладать никакой практической точностью в реальной жизни.

Однако, нам нужно как раз и понять, какую ожидать точность от модели, когда мы выведем её в интеграцию с настоящей системой, где она будет работать уже на неизвестных ранее данных, которые должны быть похожими на настоящие, но *ими не являются*!

Для того, чтобы нам это понять, всю первоначальную выборку разбивают обычно на две части: это обучающие и проверочные данные.

Процент разбиения является относительно произвольным, то есть, когда данных очень много, то можно соотнести 60/40; когда данных не очень много, то разбиение может быть выражено соотношением 85/15.

В представленном пособие разбиение будет соотнесено в отношение 80 на 20. Критичной точности нет, самое главное, чтобы на одном и том же наборе данных, когда вы сравниваете модели машинного обучения на этом наборе данных, у вас разделение данных было *одинаковым*. То есть, если в одной модели машинного обучения разбиение было в соотношение 80/20, то в конкурентной модели (которую вы реализуете для выбора лучшей модели из

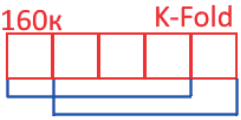
нескольких) соотношение должно быть аналогичным. Запомните, что неправильно сравнивать точности моделей на разных соотношениях.

Эти цифры соотношений (в нашем случае 80/20) выбраны для минимизации ошибок как первого, так и второго рода. В первом случае, чтобы мы могли достаточно хорошо обучить модель (чтобы она получила достаточно большое количество исходных данных) и, во втором случае, чтобы мы могли найти все ошибки этой модели, чтобы узнать, где она училась недостаточно хорошо.

Ошибки первого и второго рода мы устранили. Проверочная часть нашей выборки (в нашем случае 20 % от исходного набора) откладываются в сторону и никак не используется. А вся модель получает на входе только вот эти самые 80 % выборки и уже по ним строит свой черный ящик функций.

Однако, в ходе обучения модели, нам часто нужно провести оптимизацию гиперпараметров (подробнее в следующих главах), чтобы улучшить модель. И для того, чтобы модель улучшить на данных, выполняют следующий тип разбиения, называемый перекрестной валидацией.

K-Fold – валидация подразумевает, что, например, проводя разбиение на 5 частей мы обучаем модель на первых четырех частях и проверяем на пятой. Потом мы обучаем модель на последних 4 частях проверяем на первой. Аналогично операция повторяется столько раз, на сколько частей разбивали данные.



Получается, что мы модель последовательно обучили на всей нашей обучающей выборке и последовательно проверили на каждой из этих частей. Это исключает возможность недообучения модели и позволяет нам понять, насколько наша модель хорошо работает для обучающей выборки. Это очень распространенный пример при оптимизации параметров.

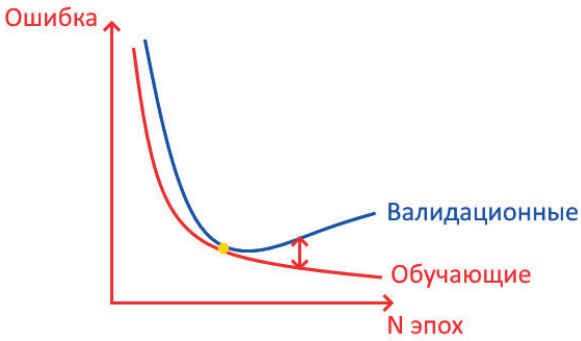
Иными словами, K-Fold – разбиение используется для оптимизации модели относительно себя самой, в то время как отложенная выборка (80/20) необходима для эталонной оценки конкурирующих моделей и сравнения по метрикам точности.

При работе с нейросетями используется ещё один тип разбиения (обучающая и валидационная выборка). Также обычно в соотношение 10/90, 80/20 и так далее валидация/обучение.

Обучение Валидация
80/20
640к 160к

Нейросети достаточно склонны к переобучению и нам нужно понять, в какой момент требуется остановить обучение, чтобы избежать переобучения. Это выполняется за счет проверки на валидационном наборе.

Для этого мы оставляем валидационный набор без изменений (160.000 значений). Каждую эпоху нейросеть обучаем на наборе в 640.000 значений и потом проверяем результат работы на этом валидационном наборе (160.000). Относительно качества работы нейросети на валидационном наборе уже можно уверенно сказать, есть ли у нас переобучение или нет.



Из графика как раз и можно найти точку переобучения модели. Момент переобучения можно вовремя «поймать», когда разница между валидационной проверкой и обучающей проверкой начнет расти. В этот момент обучение модели следует остановить.

Эти методы разбиения данных используются в моделях машинного обучения.

Подытоживая вышесказанное, отметим ключевые моменты:

1. Во всех моделях данные разделяем на обучающие и проверочные.
2. Если нам нужно провести оптимизацию гиперпараметров модели, то обычно используют перекрестную кроссвалидацию. Если параметров очень много, то применяется сетка значений. Иногда используется не перекрестная валидация, а просто обучают на обучающей выборке модель с одними параметрами, модель со вторыми параметрами, модель с третьими параметрами; затем только сравнивают модели «один к одному», в случаях, если у нас немного сравнений.
3. Однако, в сложных моделях, где присутствует градиентный бустинг, случайный лес и так далее (о них подробнее в следующих гла-

вах), то K-Fold – разбиение (перекрестная валидация) очень часто используется.

4. Для нейросети дополнительно обучающую выборку разбивают на две части для того, чтобы на каждой эпохе обучения нейросети дополнительно ее валидировать и избежать переобучения.

ОПТИМИЗАЦИЯ ГИПЕРПАРАМЕТРОВ

При обучении модели машинного обучения достаточно много времени уходит на оптимизацию гиперпараметров модели.

Почему они называются гиперпараметрами? Потому что сами исходные данные, которые мы загружаем в модель, они называются параметрами. Так сложилось, что независимые переменные, некоторые параметры, на основании которых мы предсказывал решение, принято именовать именно так.

И модель машинного обучения делает как раз оптимизацию этих параметров, чтобы выдать максимально точный ответ. У нас есть заданный ответ, у нас есть параметры, которые должны соответствовать этому ответу. И мы оптимизируем эти параметры, чтобы выдать максимально точный ответ.

Поэтому оптимизация гиперпараметров – это работа самой модели машинного обучения. Однако, чтобы нам модель машинного обучения сделать более точной (улучшить алгоритм) и производят процесс оптимизации гиперпараметров.

Оптимизация гиперпараметров модели – достаточно сложный процесс и в ряде случаев он еще и трудоемкий. У нас есть несколько дополнительных весов (или дополнительных множителей в случае с линейной регрессией). Например, уравнение линейной регрессии выглядит следующим образом:

$$y = \sum_{i=1}^{n} a_i x_i + \alpha \sum_{i=1}^{n} |a_i|.$$

Коэффициент a_i находится по методу наименьших квадратов. То есть некоторые уравнения, позволяющие нам найти эти коэффициенты.

После этого наше выражение оно уже наилучшим образом оптимизирует наши данные под нужные решения на основании модели линейной регрессии. Однако, мы эту модель можем улучшить, если введем некоторые ограничения, например, на веса, которые будут использоваться при решении, чтобы эти веса были чуть менее точные, однако смогли лучшим образом обобщать исходные данные. Мы получим, возможно, решение с чуть большей ошибкой, однако, оно будет обладать большей обобщающей способностью.

В поиске этого коэффициента (α и есть как раз гиперпараметр модели) и состоит задача оптимизации гиперпараметров.

Существует два основных подхода к оптимизации:

1) жадный поиск (поиск по сетке);
2) поиск случайный.

Жадный поиск

Например, у нас есть параметр α, β, γ.

α: 0,001 ----- 1
β: 0,1 ----- 100
γ: 5 ----- 50

Таким образом, у нас есть некоторая сетка. То есть, даже если мы зададим некоторый шаг (хотя бы 0,005 – не очень большой), то по альфе мы получаем 200 значений, аналогично по бэта и гамма:

α :	0,001	1	200
β :	0,1	100	* 200
γ :	5	50	* 40

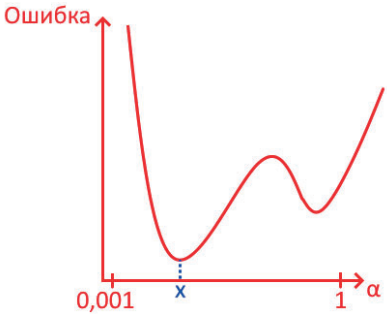
Перемножая все значения, мы получим достаточно большое пространство этих самых параметров. Здесь получается 200, умноженное на 200 и ещё на 40, то мы получаем более полутора миллионов возможных гиперпараметров модели.

И это еще они не очень точны. Каждое значение из этих полутора миллионов наборов соответствует какому-то кортежа гиперпараметров в этом пространстве гиперпараметров (лучшей будет некоторая комбинация из трех).

Поиск по сетке говорит, что мы можем взять логарифмическую шкалу для укрупнения сетки. Например, таким образом сетку можно укрупнить до 4*4*4, то есть 64 параметров. Так как мы находим какой-то определенный набор из сетки возможных параметров, то название для метода весьма удачное.

Поиск выполняется при помощи перекрестной валидации, разбивая выборку на несколько частей и проверяя значения результатов работы модели на данном наборе гиперпараметров.

Что касается ошибки и как в целом выглядит функция значения ошибки (например, у нас есть параметр альфа, меняющийся от 0,001 до 1).



В точке минимальной ошибки и будут лучшее значение гиперпараметра. Точно найти точку сразу сложно, мы перейдем сначала в некоторую окрестность, а потом итеративным подбором мы эту точку найдем.

Соответственно, нам важно, чтобы у нас исходное пространство гиперпараметров было достаточно мелкой сеткой накрыто. Далее находим некоторый локальный минимум и его исследуем на предмет того, как ведет себя значение ошибки от гиперпараметров на этом наборе.

Случайный поиск

Случайный поиск оптимального набора гиперпараметров мало чем отличается от сеточного.

Мы из всего набора помещаем в случайные места значения, также мы можем поместить в разные места 64 разных точек. Прозондировав пространство гиперпараметров, мы получаем некоторую информацию о кривой ошибок нашего пространства гиперпараметров и пытаемся выявить, в зависимости от помещенных зондов, локальный минимум ошибки на всем пространстве.

Однако, методы не лишены недостатков: при сеточном поиске можно попасть в некоторый локальный минимум, а про существование глобального минимума можем так и не узнать.

Тоже самое возможно при случайном поиске.

Однако, достаточно мелкая сетка гиперпараметров, в должной степени гарантирует, что вы найдете оптимальный набор гиперпараметров. Но, оптимизация гиперпараметров – это обычно последнее, что делают при работе с обучением модели.

Потому что оптимизация гиперпараметров не способна многократно улучшить качество. Её применяют для «настройки» модели на финальной стадии, обычно улучшая модель на 10–20 % (максимум в два раза).

При наблюдении больших проблем с точностью не старайтесь сосредоточенно улучшать гиперпараметры, потому что, скорее всего, необходимо выбрать другую модель, либо взять ансамбль моделей.

НЕДООБУЧЕНИЕ И ПЕРЕОБУЧЕНИЕ

При обучении модели машинного обучения мы достаточно часто сталкиваемся с понятиями недообучение и переобучение.

С понятием недообучение можно логически разобраться самостоятельно, то есть, когда обученная модель машинного обучения на проверочной (тестовой) выборке данных являет нам точность, которая ниже нужной нам (ожидаемой).

Недообучение может возникать по нескольким причинам, например, мы не провели оптимизацию гиперпараметров модели; в том числе могла быть выбрана неверная функция ошибки для нашей модели; также мы могли слиш-

ком рано остановить модель обучения. И, как наиболее частая ошибка, – неверный выбор самой модели машинного обучения.

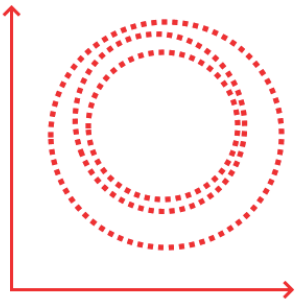
Основные причины недообучения:

1. Мало эпох (ранняя остановка в обучение).
2. Неоптимальные гиперпараметры.
3. Неверная функция ошибки.
4. Неверная модель.

Теперь перейдем к переобучению. Казалось бы, что там может произойти, если нам мало обучения, то можно добавить больше эпох. При неверной функции ошибки мы выберем другую функцию, она даст еще лучше результат. И так далее.

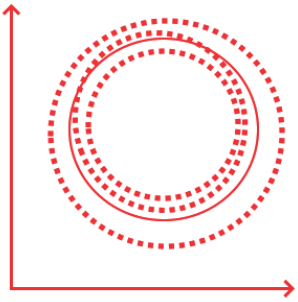
В чем состоит проблема переобучения можно рассмотреть на небольшом, игровом примере. Он проиллюстрирует к чему может привести переобучение.

Допустим, у нас существует ряд точек окружности:



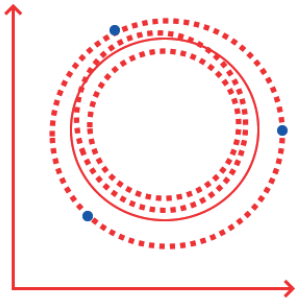
Мы хотим, чтобы в результате оптимизации работы нашей функций получить предсказание некоторой окружности.

Разумеется, что она не сможет охватить все точки, если мы хотим нарисовать всего одну линию. Однако это будет некоторая оптимальная линия:

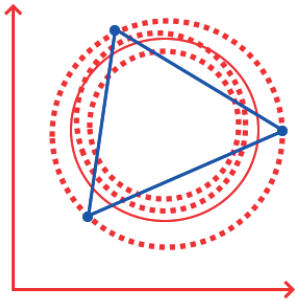


Мы хотим в результаты работы нашей модели машинного обучения получить именно окружность. Это будет идеальный результат нашей модели.

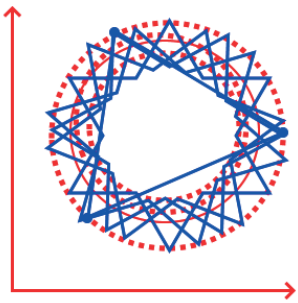
Однако, модель машинного обучения про это ничего не знает. Для того, чтобы она справилась с задачей, мы передаем ей на вход некоторую выборку. Например, передадим модели три основных контурных точки, чтобы модель дорисовала окружность:



Однако модель может сразу же определить в них треугольник и дорисовать треугольник:



В результате, если мы загрузим в модель очень много разных данных, то в конце концов, по форме будет примерно похожая на окружность (точнее повторяющая её контуры фигура):

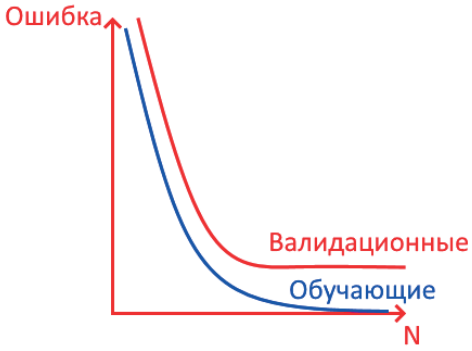


Это будет наша самая лучшая модель машинного обучения. Как вы понимаете, вид этой фигуры крайне плохо напоминает окружность. Но точность модели будет соответствовать заявленным 100 %. Иными словами, модель приспособится под наши исходные данные и не сможет решить задачу с другим радиусом окружности, так как будет заточена только под единственный набор, на котором она «переобучилась».

Поэтому процессе обучения модели машинного обучения важно отследить как раз тот момент, когда у нас модель переобучается, а когда она ещё недообучилась. В случае с нейронными сетями это сделать немножко проще.

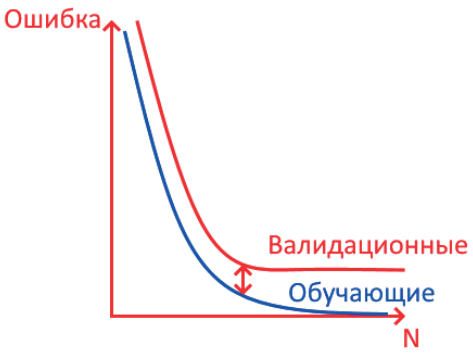
В случае с классификационными или линейными моделями это сделать немножко сложнее, но везде помогают работать с этим ансамбли моделей. Все прикладные тонкости работы с конкретными моделями для избегания проблем переобучения и недообучения мы будем рассматривать в следующих главах.

Для решения проблемы вводится процедура проверки на валидационной выборке. Нам необходимо проверять зависимость данных от ошибки не только на обучающей выборке, но и на валидационной:



Валидационная выборка – это извлеченные несколько точек из обучающей выборки, которые не участвуют в обучении. При проверке на валидационной выборке, в некоторый момент переобучения ошибка будет, как минимум, переставать падать, а в большинстве случаев вновь начинать увеличиваться. Этот момент необходимо отследить позже и, выбрав значения параметров в точке с наименьшей ошибкой по валидационной выборке взять за основу.

И как только начинает увеличиваться зазор на графике (аналитически – это разница ошибок – значений функции – при одинаковых значениях аргумента) между валидационной и обучающей выборкой мы можем уверенно сказать, что у нас есть переобучение модели:



После этого момента модель дальше уже не просто не нужно обучать, а вредно, поскольку она начнет ухудшать свою предсказательную сил.

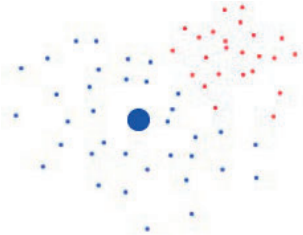
Во всех моделях машинного обучения необходимо помнить о разнице двух типов ошибок, а также о том, в какой момент нужно остановить обучение модели. Либо, в случае ансамблирования моделей, добавить модель в ансамбль на данном этапе, проводя обучение уже внутри ансамбля, ориентируясь на разрыв между ошибками.

СМЕЩЕНИЕ, РАЗБРОС И ОШИБКА ДАННЫХ

Продолжая разговор про машинное обучение, необходимо упомянуть про такие понятия как смещение и разброс. И относительно них, естественно, некоторую ошибку или погрешность данных.

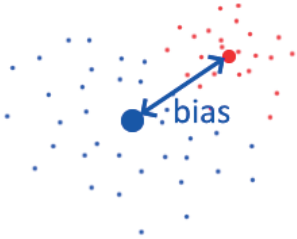
Давайте посмотрим, как может выглядеть смещение или разброс, что это вообще такое. Предположим, что у нас есть некоторая цель (например, некоторое целевое значение или класс). И мы хотим, чтобы наша модель попала ровно в него. Но модель в него не попадает, она дает предсказания примерно следующим образом:

На самом деле, ошибается не только модель. Она ошибается потому, что сами наши данные тоже не расположены кучно и они могут быть тоже как-то отделены от самого центра:



Центр – это некоторое идеальное состояние данных, некоторое идеальное предсказание, которое данные должны принять. Это может быть либо среднее значение, либо некоторое ожидаемое значение. В любом случае это целевое значение, которое мы хотим от модели машинного обучения получить.

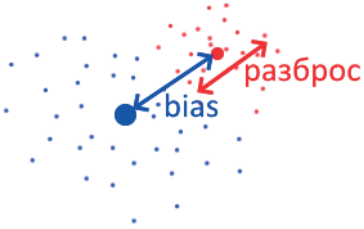
Обычно оно, во-первых, в данных отсутствует. Во-вторых, модель машинного обучения смещается относительно этого предсказания. Смещение – это расстояние между математическим ожиданием данных и математическим ожиданием самой модели. Принято именовать его понятием bias.



С точки зрения практики, смещение модели – это когда мы между средним предсказываемым значением и средним значением наших данных (тестовых данных) вычисляем значение расстояния.

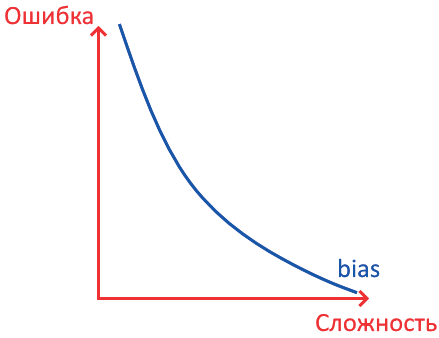
Нет идеального в моделях машинного обучения, поэтому задача инженеров – построить модель, которая будет иметь небольшую величину смещения. Однако смещение сильно зависит от характера самой модели.

Кроме смещения есть ещё разброс (облако значений):

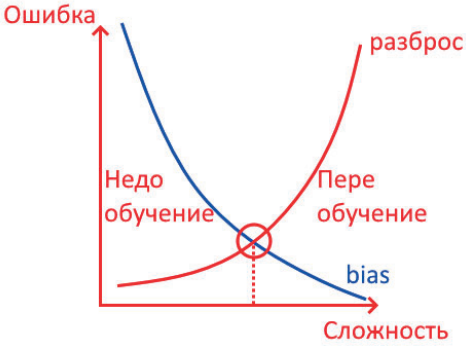


Разброс – это отклонение предсказаний от математического ожидания самой модели.

Если мы попробуем построить какую-то зависимость по сложности и ошибке предсказания, то получится примерно такой график:



Получается, что смещение ***уменьшается*** для более сложной модели. Более сложная модель становится всё более и более близкой к истинному среднему, истинному предсказываемому значение (ожидаемому предсказываемому значению). В целом, смещение с повышением сложности увеличивается. Однако, разброс ***увеличивается*** для более сложной модели. Поэтому необходимо найти некоторую оптимальную область при построении модели:



На фоне разброса и смещения также вводят некоторую ошибку, называемую ошибкой данных. Ошибка данных складывается из статистического шума и погрешности.

Нужно понимать, что статистический шум – это некоторые не очень характерные значения или ошибки измерения.

Статистический шум – случайность измерения, то есть измерения у нас находятся в каких-то границах (математическое ожидание плюс минус квад-

ратный корень из дисперсии величины). Отклонения в одну-другую сторону относятся к статистическому шуму.

Кроме статистического шума, на эту ошибку измерения накладываются еще и погрешность. В итоге, у нас данные могут быть достаточно сильно «болтаться».

Соответственно, ***ошибка модели не может быть меньше ошибки данных***.

Это тоже нужно всегда иметь в виду и ориентироваться в том числе на ошибку данных при работе с моделью машинного обучения.

И еще один важный момент: для борьбы с большим разбросом (или в целом разбросом одной конкретной модели) делают ансамбль этих моделей – ансамбль бэггинга. Когда у нас много однородных моделей, то необходимо их немного случайным образом инициализируем (внося тем самым случайность) и просто усредняем. Как мы помним, ошибка среднего в $\sqrt{N}$ раз меньше, чем ошибка одного.

Ансамбль бэггинга уменьшает разброс в $\sqrt{N}$ раз. Взяв 16 моделей, ошибка тем самым уменьшится в 4 раза!

Для уменьшения смещения применяется другая математическая хитрость: вводится ансамбль стекинга, который уменьшает смещение моделей.

ИСПОЛЬЗОВАНИЕ HDF

После обучения модели машинного обучения результаты обычно отгружаются в каком-то виде или в формате. Наиболее часто используемый формат для выгрузки результатов является HDF (Hierarchial Data Format) и файл HDF4 или HDF5.

Он является достаточно расширяемым с одной стороны и бинарным с другой.

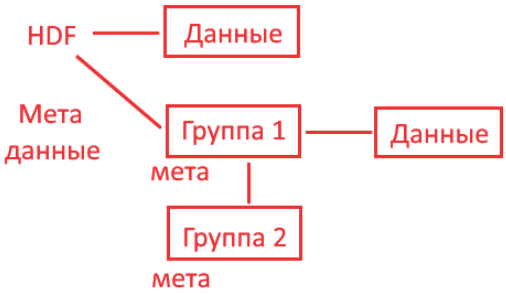
Бинарность позволяет нам очень компактно хранить данные быстро получать к ним доступ.

Расширяемость позволяет нам хранить в принципе любые объекты данных.

У формата файла HDF существует некоторый корень. Естественно, у него есть метаданные. Дальше у корня могут быть данные; данные в корни могут приходить. Данные могут содержать какую-то группу. При этом группа может также содержать ещё одну группу или целый ряд групп.

Соответственно, мы можем таким образом организовать хранение произвольной информации в виде специфического дерева. Естественно, что у каждой группы могут быть метаданные. К данным метаданные не привязаны.

Но обычно, если нужно привязать метаданные к информации, то организуют группу с метаинформацией и в неё кладут данные.



HDF отлично подходит как для хранения результатов предобработанных данных: мы в одном месте собираем все очищенные данные, нормализованные и с заполненными пропусками.

Кроме этого, формат HDF используется для того, чтобы модель пересчитать, оптимизировать, либо провести ансамблирование моделей на одном и том же наборе подготовленных данных.

Также достаточно удобно в HDF складывать разбитые на обучающие и проверочные данные. Один раз разбив данные, легко воспроизвести тренировку и обучение различных моделей на различных устройствах (или даже различных серверах) *по одному и тому же разбитому набору данных*.

Это существенно, потому что модель, в зависимости от разбиения данных, может выдавать разные результаты.

HDF является гораздо более лучшим вариантом, чем формат *.csv или даже база данных.

С базой данных он может быть сравним по скорости чтению данных и объему потребляемой памяти: для больших данных (big data) возможно использование специализированных хранилищ. HDF подходит для десятков – сотен гигабайт исходных данных.

Так же HDF отлично подходит для:

1. Хранение обработанных данных.
2. Хранение описания модели (весов нейросети).
3. Получение срезов данных/гиперкубов (выбирать срез не только по дате, объему и так далее, но и по набору параметров этих данных. Следует отметить, что по скорости работы операций чтения/записи структура файла HDF не уступает базам данным, НО не требует развертки определенного, как правило, очень тяжеловесного программного обеспечения и не потребляет столько аппаратных ресурсов).

Поскольку в HDF существует встроенная процедура сохранения слепка в адресе информации, то файл такого расширения заменяет индексируемую базу данных. Для индексации баз данных требуются существенные ресурсы процессора, кроме того, индексация негативно отражается на продолжительности работы твердотельных накопителей.

При сохранении данных формат HDF сравним с другими файлами или форматами по скорости загрузки.

При чтении данных чтение осуществляется во много раз быстрее.

Поскольку формат имеет иерархическую структуру, то он часто используется для хранения всей модели в целом и хранение весов нейросетей в том числе (HDF – это наиболее распространённый формат для хранения весов нейросетей).

Формат позволяет оперативно и очень экономно (с точки зрения аппаратных ресурсов) выполнять срезы данных (поднаборы в исходных наборах данных).

Часть 2
МЕТРИКИ И МОДЕЛИ ОБЩИЕ

МЕТОД МАКСИМАЛЬНОГО ПРАВДОПОДОБИЯ

Основополагающая идея метода максимального правдоподобия является на самом деле фундаментом в ряде моделей (наиболее простых моделей) машинного обучения, поэтому логично начать рассмотрение моделей, метрик, работы с моделями с метода максимального правдоподобия.

Давайте рассмотрим такой небольшой пример, и мы на нем посмотрим в чём заключается метод и как нам с ним можно работать. Например, предположим, у нас в группе по машинному обучению учится 10 человек: из них 6 девушек и четыре парня. Если мы предположим, что у нас есть некоторое соотношение между всеми девушками и парнями во внешнем мире (среди всех 8000000000 человек) и нам было бы интересно узнать, сколько должно быть девушек среди тех 8000000000, чтобы когда мы набрали в группу 10 случайных человек за этих 8000000000 у нас случилось, что в ней будет 6 девушек и 4 парня. А поскольку это событие случилось, то скорее всего, в этом заключается метод максимального правдоподобия, скорее всего вероятность этого события должна быть максимальной из всех возможных.

Как оценить вероятность события? Она оценивается через функцию правдоподобия.

6 девушек, 4 парня

$$Fn = P_1 * \ldots * P_{10} \longrightarrow max$$

Вводится функция правдоподобия, поскольку мы считаем, что события у нас независимые, то вывести её достаточно просто: она будет равна произведению вероятностей получить каждого участника группы из всех 8000000000 (то есть мы выбираем одного участника, смотрим в данном контексте на его пол, выбираем второго участника, смотрим на его пол, и у нас есть некоторое произведение вероятности, что вот именно такой состав группы у нас случился). Функция правдоподобия должна при этом быть максимальной, то есть поскольку у нас уже есть свершившееся событие, вероятность у него 100 % (или единица), то очевидно, что вероятность, описывающая это событие, стремится к своему максимуму, для того чтобы у нас что-то случилось, чтобы поняли, чтобы могли, например, корректно описать всю генеральную совокупность на 8000000000 человек (все данные, из которых мы взяли небольшую часть – выборку, и функция правдоподобия имев максимальное правдоподобие, позволяет оценить всю нашу генеральную совокуп-

ность, все данные, из которых мы выбираем значение. Зачем нам нужно оценивать все данные? Потому что в модели машинного обучения мы учимся на небольшом участке данных и оценив, что у нас за пределами обучающих данных находится, мы можем очень хорошо саму модель машинного обучения сделать, чтобы усилить её предсказательную силу, снизить ошибку и так далее, так далее.

На практике, взглянув за пределы данных, которые у нас есть в модели машинного обучения, мы выполним половину работы, если мы узнаем какие-то простые взаимосвязи данных, которые там присутствуют. Так вот в случае с бинарной классификацией, метод максимального правдоподобия (когда произведение устремляем к максимуму), как раз и даёт, что у нас во всей генеральной совокупности, во всём населении земли, откуда мы взяли 10 человек, соотношение девушек и парней будет как раз 60/40. Соответственно, вероятность того, что случайно выбранный человек окажется девушкой – 60 % (судя по нашей выборке, потому что она получилась именно такой), исходя из максимизации функции вероятности, которая зависит от некоторого неизвестного параметра, описывающего генеральную совокупность (нашего населения), все наши данные, из которых мы получили выборку, мы, в случае с бинарной классификацией можем с высокой точностью предсказать пол человека.

Отсюда следует понятный и логичный вывод, что сколько мы взяли в выборки одних элементов, сколько других элементов, то таковой, скорее всего, является вся генеральная совокупность, то есть такие наши данные и есть. Он напрямую следует из максимизации функции правдоподобия и метода максимального правдоподобия. Однако, для максимизации функции правдоподобия, произведение тяжело дифференцировать, тяжело искать его максимума, поэтому вводят логарифм функции правдоподобия как сумму логарифмов всех вероятностей получить данные значения в группе выборки, и уже для суммы достаточно просто искать максимум, потому что, как минимум, её можно продифференцировать уже гораздо проще: у нас будет простое отношение 1/p.

$$Fn = P_1 * ... * P_{10} \to \max$$
$$\ln Fn = \ln P_1 + ... + \ln P_{10}.$$

Таким образом, максимизация функции правдоподобия и метод максимального правдоподобия для бинарной классификации даст нам просто значение: в данном случае 0,6, то есть 60 % девушек у нас, скорее всего, встретится во всех данных.

Для линейной регрессии, когда у нас существуют некоторые линейные зависимости между предсказываемой, зависимой величиной, и независимыми параметрами (рисунок NNN).

$$y = a * x + b.$$

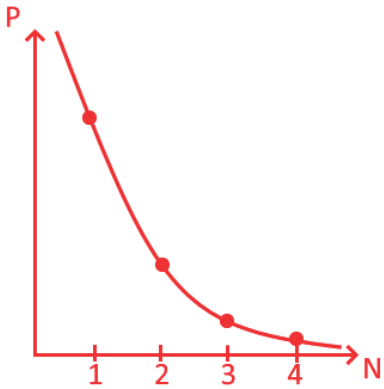
В этом случае, если мы приведём функцию вероятности, функцию нашего правдоподобия, то вероятность получить именно эту выборку, которая описы-

вается линейным законом распределения, мы как раз и получим коэффициенты, то есть дифференцирование – это уравнение для метода линейной регрессии, она даст нам как раз коэффициент a и b, – то, что и называется методом наименьших квадратов. Это является прямым следствием метода максимального правдоподобия, только для уравнения линейной регрессии. Метод максимального правдоподобия также применяется в методе ближайших соседей.

Однако существует ряд проблем: если выборка ограничена, и мы выбрали оттуда уже некоторое количество элементов, то вероятности уже не будут равны между собой (первой и десятой, допустим).

Чтобы оценить зависимости событий вводятся апостериорные вероятности, и эта проблема решается в модели наивного Байеса. Модель Байеса также базируется на методе максимального правдоподобия.

Также еще один важный вывод, следующий из метода максимального правдоподобия. Если нам следует оценить вероятность некоторого события, которое случается через регулярные промежутки времени (покупки в магазине, поход на работу). Тогда существует специальная экспоненциальная функция, которая позволяет определить, что с течением времени вероятность события падает, либо количество их через равные промежутки времени уменьшается по экспоненциальному закону:



То, что параметр экспоненциального распределения равен выражению скорости угасания вероятности при увеличении числа событий выводится именно из метода максимального правдоподобия. Описывается он средним значением лямбда. Этот параметр может быть выведен из метода максимального правдоподобия, если мы допустим, что данные описываются экспоненциальным законом распределения вероятности.

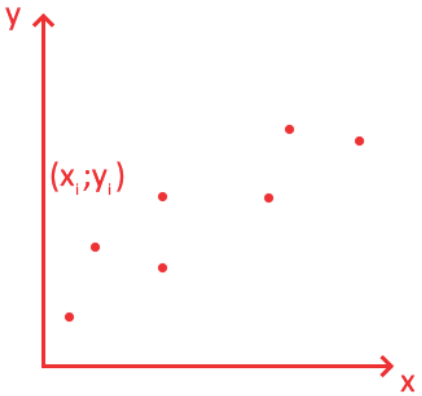
Подводя итог вышеописанному, метод максимального правдоподобия может быть сформулирован кратким выражением: функция правдоподобия при выборке из 10 человек (пример в начале) принимает все возможные ситуации, но её максимум находится только в нашей ситуации, поскольку именно

эта ситуация случилась. Исходя из того, что оно случилось появляется возможность посчитать, при каких значениях неизвестного распределения генеральной совокупности, неизвестная доля (не зная точно, какой процент 10 человек составляют от генеральной совокупности) может быть оценена из функции правдоподобия и её максимизации. Поэтому окажется, что для бинарной классификации это точное значение – доля; для ближайшего соседа – также доля (наиболее частое значение рядом); для линейной регрессии – коэффициент линейной регрессии; для наивного Байеса (с уточнением апостериорной вероятности) – формула вероятности наивного Байеса; для экспоненциального распределения – среднее значение, описывающее функцию угасания.

МЕТОД НАИМЕНЬШИХ КВАДРАТОВ

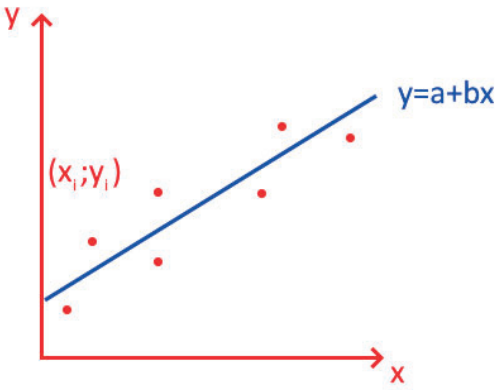
Метод наименьших квадратов является одним из базовых методов машинного обучения и применяется для оценки ошибок моделей не только линейных. Он очень простой и позволяет вам найти точные решения коэффициентов, которые минимизируют ошибку. Логично начать разговор про модели машинного обучения именно с метода меньших квадратов.

Предположим, у нас есть некоторое количество данных (представлены как точки).

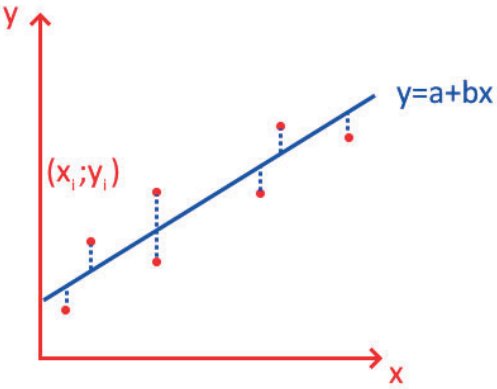


Начнем искать уравнение этой линии, которое минимизирует ошибку в наших данных. То есть уравнение, которое описывает по линейным законам наши данные, при этом является наиболее оптимальной линией, которая так расположена.

Линия практически прошла все точки и теперь нам осталось найти расстояние от всех точек до этой линии, найти минимум этого расстояния. Минимум этого расстояния даст нам соответствующие коэффициенты *a* и *b*.



Давайте это сделаем без лишних математических выкладок, просто посмотрим, что у нас есть опущенные перпендикуляры и в этом расстоянии у нас будут только разные *y*, нам нужно найти только разницу по *y*.



Запишем теперь сумму расстояний:

$$\sum_{i=1}^{n}\left(y-y_i\right)^2=\sum_{i=1}^{n}\left(y_i-a-bx_i\right)^2.$$

Для каждой точки мы вычисляем ее предсказанное значение, то есть линейная регрессия – это некоторые предсказания, вычисляемые для *y* по аргументу *x*.

Чтобы понять ошибку этой модели, достаточно вычесть из фактического значения значение нашей модели. Получая значение ошибки, возводим её в квадрат – это необходимо для того, чтобы знак никуда не девался. Это стандартная процедура для нахождения расстояния.

И теперь, если мы скажем, что функция должна стремиться к минимуму (быть минимальной из всех возможных), то это даст нам точное значение коэффициента *a* и *b*.

$$\sum_{i=1}^{n}\left(y-y_i\right)^2=\sum_{i=1}^{n}\left(y_i-a-bx_i\right)^2\to\min.$$

Коэффициенты находятся дифференцированием. Отметим, что линейная регрессия без дополнительных ограничений на веса, предполагает точное решение, которое уже является минимумом ошибки линейной регрессии.

Коэффициенты линейной регрессии рассчитываются следующим образом:

$$b=\frac{\overline{xy}-\bar{x}*\bar{y}}{\sum_{i=1}^{n}x_i^2-\overline{x^2}}.$$

$$a=\bar{y}-b\bar{x}.$$

Таким образом, данные коэффициенты для линейной регрессии являются точным решением.

Метод наименьших квадратов может использоваться не только для линейной агрессии. С помощью него мы можем оценивать ошибку любой модели, он говорит лишь о том, что мы находим расстояние (когда квадрат расстояния и все наши ошибки, которые заключаются в сумме квадратов расстояния между данными и значениями – исходными значениями данных и предсказанными значениями данных). Собственно, наши квадраты, которые мы минимизируем далее по методу наименьших квадратов. В случае линейной регрессии это дает нам дает точный ответ. Во всех остальных случаях это дает нам оценку (очень точную оценку), но саму модель нам не реализует, а выдает исключительно оценку ошибки этой модели.

АППРОКСИМАЦИЯ ПРОПУСКОВ В ДАННЫХ

Методы аппроксимации данных относятся более к вычислительной математике и не совсем к машинному обучению.

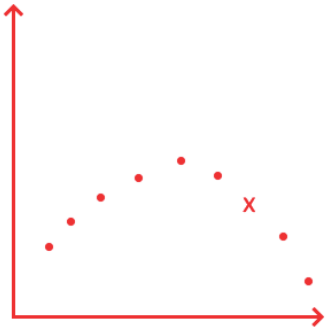
Однако, само машинное обучение является результатом работы, решением задачи по экстраполяции данных, то есть, когда у нас есть некоторый набор данных, он ограничен некоторыми пределами (временными пределами, пределами значений и другими) и нам нужно предсказать, какие у нас могут быть данные за пределами этих ограничений. Например, у нас есть данные погоды, и мы хотим предсказать на погодные условия два года вперед – это типичная задача экстраполяции и она методами вещественной математики не решается, а разрешима только методами машинного обучения.

Выделяют два разных вида аппроксимации данных:

1. Экстраполяция.
2. Интерполяция.

Интерполяция говорит, что у нас есть некоторые данные и мы хотим получить значение «между». В машинном обучении интерполяция используется для заполнения пропуска в данных и, возможно, для получения каких-то дополнительных серий данных.

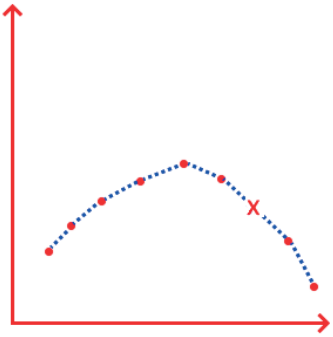
Например, у нас есть некоторый ряд значений. Рассмотрим некоторый график и тут присутствует пропуск:



Тут нет значения. Мы могли бы каким-то образом заполнить этот пропуск, нам нужно что-то наиболее характерное в эти данные вставить и при этом не использовать машинное обучение, чтобы заполнить пропуски в данных для машинного обучения. Следовательно, нужно использовать какие-то более простые методы для решения этой задачи.

Одним из наиболее простых методов интерполяции данных является линейная интерполяция данных. Линейная интерполяция строится ровно по двум точкам, соединяя эти две точки в один общий отрезок.

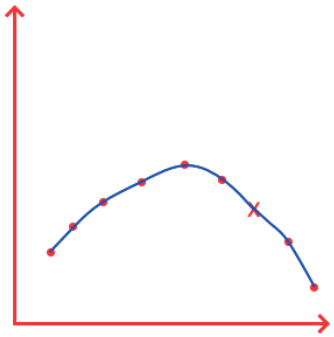
Представим результат линейной интерполяции данных.



Линейная интерполяция позволяет очень просто построить линию и найти исходную точку (коэффициент). Линейная интерполяция простая, понятная, будет вести себя понятным и предсказуемым образам. Именно поэтому её применяют, когда данных немного и эти данные как-то могут быть сильно разнесены друг относительно друга.

В большинстве случаев, когда пропусков становится очень много, то использовать линейную интерполяцию становится «невыгодно» в том плане, что она даст менее точное значение. Поэтому чуть более улучшенной версией линейной интерполяции является кубическая интерполяция.

Кубическая интерполяция дает гладкую функцию во всех точках – это называется сплайн.



Обратите внимание, что в нашей точке пропуска кубическая интерполяция пройдет чуть-чуть ближе к исходному пропущенному значению, потому что она лучше повторит характер зависимости, которая у нас есть в исходных данных.

Поэтому, когда у нас пропуски между данными небольшие (то есть они не затрагивают какие-то экстремумы, точки максимума/минимума), используют кубическую интерполяцию.

Когда данные сильно расходятся, между ними существуют большие пропуски, то лучше всего использовать линейную интерполяцию.

СРЕДНЕКВАДРАТИЧНАЯ ОШИБКА

Среднеквадратичная ошибка является достаточно распространенной метрикой при измерении качества работы линейных моделей, то есть как мы вычисляем по методу наименьших квадратов коэффициент регрессии, так мы можем, используя этот же метод, посчитать ошибку.

Собственно, она будет минимальной, но нам просто нужно понять и оценить её. Или, допустим, сравнить с ошибкой работы других моделей. Чтобы нам что-то измерить, нам предварительно нужно эту ошибку посчитать.

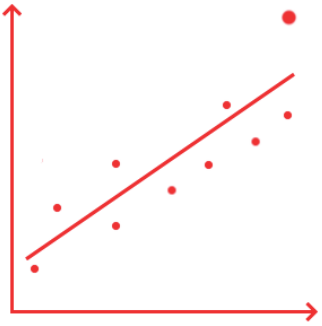
Так вот среднеквадратичная ошибка является базовой и самой простой оценкой. Она обозначается RMSE (Root-mean-square deviation) и исчисляется как корень квадратный из MSE (Mean squared error).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y - y_i\right)^2.$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y - y_i\right)^2}.$$

Понятие ошибки – это некоторое усредненная ошибка по расстоянию. Метрика очень хороша тем, что она дифференцируема, по ней можно двигаться – если у нас есть какая-то сложная, нелинейная функция, то мы можем использовать эту ошибку, чтобы идти к минимуму функции.

Однако метрика плохо описывает поведение модели при больших выбросах:



При явном выбросе регрессия стремится к ошибке, но, если это выброс, то его вообще имеет смысл исключить из выборки. Неизвестность того, должны ли мы учитывать выброс или нет зависит от постановки задачи.

Так как мы в самом начале работы с моделью выбираем метрику, по которой будем оценивать результат. Поэтому среднеквадратичная ошибка может быть неправильной метрикой, по которой мы будем оценивать работу нашей модели. Потому что в случае больших выбросов, метрика себя ведёт очень плохо – модель начинает «выгибаться» и стремиться к большим выбросам.

Для того, чтобы свести влияние больших выбросов и больших ошибок, которые могут быть случайными, к минимуму, берут разность алгоритмов. Разность значений сводим на разность логарифмов и вводят методику RMSLE (RMSLE = Логарифмическая RMSE).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\log\left(y+1\right) - \log\left(y_i+1\right)\right)^2}.$$

Единственным ограничением метрики является то, что выражение под логарифмом должно всегда быть положительным. За счет того, что в формуле присутствует разность алгоритмов, то выбросы будут меньше учитываться в ошибке и модель будет строиться чуть лучше. Но опять-таки понятие лучше и хуже здесь относительное: качество всегда определяется той методикой, которую мы вводим при постановке задачи.

Если мы говорим, что будем использовать RMSE, то тогда наша модель должна учитывать *все* выбросы и считать их существенными.

Следует всегда помнить, что выбор метрики ошибки – одно из главных в реализации модели машинного обучения, так как только оценка работы нашей модели относительно методики в целом позволит понять хорошо или плохо функционирует модель, потому что модель – это набор уравнений, которые каким-то образом обрабатывают данные. Именно поэтому критерий «хорошо» вводится только за счет метрики. А среднеквадратичная ошибка и среднеквадратичная логарифмическая ошибка являются одними из наиболее простых метрик; в случае линейной регрессии данные метрики используются практически повсеместно.

МЕТРИКИ И РАССТОЯНИЯ

При оценке модели машинного обучения, кроме обычного (Евклидова) расстояния используются также и другие метрики.

При использовании метрики среднеквадратичной ошибки у нас есть некоторое Евклидово расстояние.

$$\sqrt{\sum_{i=1}^{n}\left(x-x_i\right)^2}.$$

Евклидово расстояние для трех координат – это сумма квадратов разности между всеми координатами.

Кроме Евклидова расстояния иногда, как мы уже заметили, используют квадрат Евклидова расстояния.

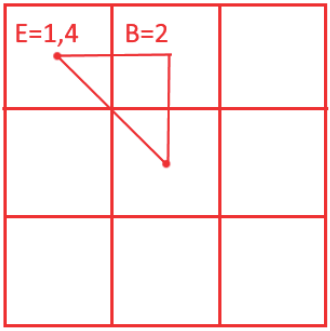
$$\sum_{i=1}^{n}\left(x-x_i\right)^2.$$

Он необходим для исключения квадратного корня. Поскольку корень квадратный – это монотонная функция, там, где у нас минимум суммы квадратов, будет минимум и у корня квадратного. Очень часто Евклидово расстояние заменяет на квадрат Евклидового расстояния, ничего при этом не теряя от метрики.

Кроме Евклидового расстояния есть ещё и так называемое L_1 – расстояние или «расстояние городских кварталов». Оно записывается как:

$$\sum_{i=1}^{n}\left|x-x_i\right|.$$

Рассмотрим разницу метрик на примере шахматной доски и расстояния между двумя точками:



Расстояние евклидово равно квадратному корню из 2, то есть примерно 1,4. А вот расстояние городских кварталов равно уже 2.

В ряде задач метрика L_1 более предпочтительна, особенно когда данные – это не числа, а ранг.

Кроме расстояния городских кварталов также есть расстояние Чебышева. В нём берется не сумма, а максимум:

$$\max\left|x - x_i\right|.$$

В выше представленной задаче расстояние Чебышева равно единице, так как минимальная разница между клетками равна 1.

Расстояние Чебышева полезно, когда получаются разные оценки. Благодаря нему можно посчитать число ошибок по каждому кортежу данных. В то время как евклидово расстояние решает проблему качества ошибки.

Также есть еще расстояние Минковского, которое на обобщает все три расстояния и записывается как:

$$\sqrt[p]{\sum_{i=1}^{n}\left(\left|x\right| - \left|x_i\right|\right)^p}.$$

Для $p = 1$ формула сводится к формуле расстояния городских кварталов, для $p = 2$ – к евклидову расстоянию. Если p стремится к бесконечности, то формула сводится к расстоянию Чебышева.

Разумеется, что в некоторых математических пакетах очень часто требуется ввести расстояние Минковского с каким-то параметром по умолчанию.

Также, в случае оценки категориальных данных, нам нужно как-то понять как у нас данные разнятся по категориям. И здесь мы вводим специальную категориальную оценку. Для этого для двух элементов данных мы считаем число разошедшихся категорий.

$$Value\left|A \neq B\right|.$$

Например, есть женщина, 26 лет, по профессии бухгалтер. Также есть мужчина, 32 года, профессия бухгалтер. Соответственно, они разошлись на две трети (на два признака из трех) и сошлись по одному признаку (профессия). В этом случае нормированное расстояние будет равно 1/3. Его можно не нормировать (тогда расстояние будет равно 2, так как по 2 признаком наблюдается расхождение).

В меньшей степени в машинном обучении применяется категориальная оценка, потому что в процессе обработки данных можно легко перевести все реальные категориальные данные в числовые. А дальше работать с числовыми данными как будет удобно, например, по расстоянию Минковского. Благодаря числовому соотнесению данных друг к другу всегда сохраняется возможность найти ошибку работы модели машинного обучения.

ЧАСТЬ ПРАКТИЧЕСКИХ НАВЫКОВ К 1–2

ПРОЦЕСС ETL

По этой теме мы разберем задачу о предсказании потреблении энергии воды и водоотведения ASHRAE.

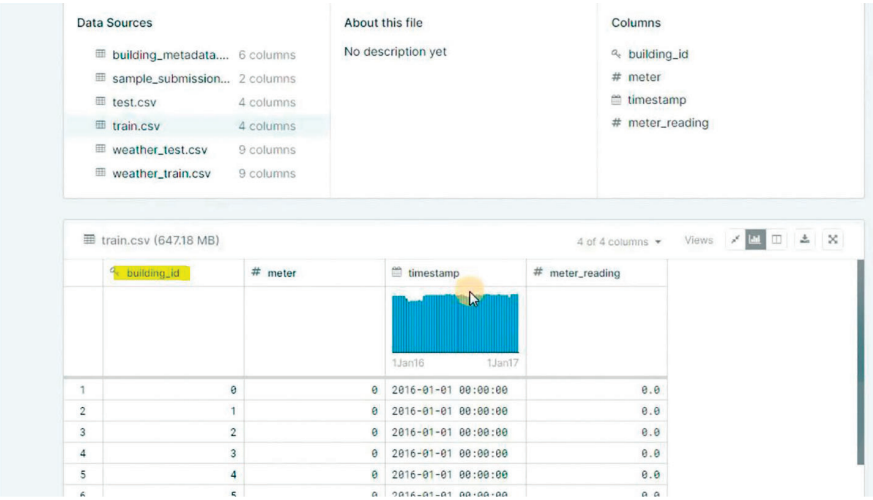
Набор данных содержит потребление примерно 2000 зданий 16 городов мира за период в один календарный год. А также данные по погоде. Требуется предсказать потребление этих зданий на полтора года вперёд с учётом известных метеорологических данных погоды.

В следующих практических уроках будут последовательно разобраны подходы к решению задачи при помощи алгоритма линейной регрессии. Методика ETL подразумевает под собой получение, очистку и объединение входных данных для дальнейшего анализа, извлечения смысла и обучения моделей.

Применительно к нашей задаче следует загрузить три набора данных:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz
2. http://video.ittensive.com/machine-learning/ashrae/train.0.0.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz

Первый набор содержит building_id:



Для него есть данные из второго набора. Первый набор содержит site_id, для которого есть данные в третьем наборе.

Затем нужно объединить все наборы данных по building_id, site_id и выполнить очистку данных от посторонних значений.

Для выполнения ячейки необходимо нажать комбинацию Ctrl+Enter. Допустим, библиотека matplotlib отсутствует на локальном компьютере, тогда Jupyter выведет ошибку:

Подключение библиотек

```
1]: %matplotlib inline
    import pandas as pd
    import matplotlib.pyplot as plt
    from matplotlib.pyplot import rcParams
    rcParams['figure.figsize'] = 16, 8
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
C:\Users\B19A~1\AppData\Local\Temp/ipykernel_8892/4194417949.py in <module>
----> 1 get_ipython().run_line_magic('matplotlib', 'inline')
      2 import pandas as pd
      3 import matplotlib.pyplot as plt
      4 from matplotlib.pyplot import rcParams
      5 rcParams['figure.figsize'] = 16, 8
```

Установка библиотеки выполняется через командную строку или прямо в контейнере пустой ячейки среды Jupyter:

```
Ввод [2]: pip install matplotlib
```

Для начала работы подключаем библиотеки:

Подключение библиотек

```
Ввод [26]: %matplotlib inline
           import pandas as pd
           import matplotlib.pyplot as plt
           from matplotlib.pyplot import rcParams
           rcParams['figure.figsize'] = 16, 8
```

Далее определимся с тем, что значат параметры в данных. То есть какие они несут в себе значения:

Загрузка данных: здания

- primary_use - назначение
- square_feet - площадь, кв.футы
- year_built - год постройки
- floor_count - число этажей

```
Ввод [27]: buildings = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz")
           print (buildings.head())
```

	site_id	building_id	primary_use	square_feet	year_built	floor_count
0	0	0	Education	7432	2008.0	NaN
1	0	1	Education	2720	2004.0	NaN
2	0	2	Education	5376	1991.0	NaN
3	0	3	Education	23685	2002.0	NaN
4	0	4	Education	116607	1975.0	NaN

Следующим шагом загружаем данные погоды:

Загрузка данных: погода

- air_temperature - температура воздуха, C
- dew_temperature - точка росы (влажность), C
- cloud_coverage - облачность, %
- precip_depth_1_hr - количество осадков, мм/час
- sea_level_pressure - давление, мбар
- wind_direction - направление ветра, градусы
- wind_speed - скорость ветра, м/с

Ввод [28]:
```
weather = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz")
print (weather.head())
```

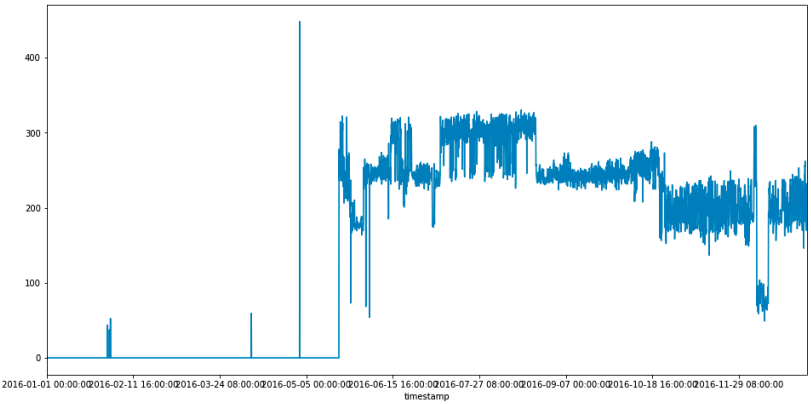
```
   site_id            timestamp  air_temperature  cloud_coverage  \
0        0  2016-01-01 00:00:00             25.0             6.0
1        0  2016-01-01 01:00:00             24.4             NaN
2        0  2016-01-01 02:00:00             22.8             2.0
3        0  2016-01-01 03:00:00             21.1             2.0
4        0  2016-01-01 04:00:00             20.0             2.0

   dew_temperature  precip_depth_1_hr  sea_level_pressure  wind_direction  \
0             20.0                NaN              1019.7             0.0
1             21.1               -1.0              1020.2            70.0
2             21.1                0.0              1020.2             0.0
3             20.6                0.0              1020.1             0.0
4             20.0               -1.0              1020.0           250.0

   wind_speed
0         0.0
1         1.5
2         0.0
3         0.0
4         2.6
```

Просмотрим данные энергопотребления здания 0:

Ввод [29]:
```
energy_0 = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.0.csv.gz")
print (energy_0.head())
energy_0.set_index("timestamp")["meter_reading"].plot()
plt.show()
```

```
   building_id  meter            timestamp  meter_reading
0            0      0  2016-01-01 00:00:00            0.0
1            0      0  2016-01-01 01:00:00            0.0
2            0      0  2016-01-01 02:00:00            0.0
3            0      0  2016-01-01 03:00:00            0.0
4            0      0  2016-01-01 04:00:00            0.0
```

Но лучше отобразить это графически:



Объединим потребление электроэнергии и информацию о зданиях по столбцу building_id (идентификатор здания):

Ввод [30]:
```
energy_0 = pd.merge(left=energy_0, right=buildings, how="left",
                    left_on="building_id", right_on="building_id")
print (energy_0.head())
```

```
   building_id  meter            timestamp  meter_reading  site_id  \
0            0      0  2016-01-01 00:00:00            0.0        0
1            0      0  2016-01-01 01:00:00            0.0        0
2            0      0  2016-01-01 02:00:00            0.0        0
3            0      0  2016-01-01 03:00:00            0.0        0
4            0      0  2016-01-01 04:00:00            0.0        0

  primary_use  square_feet  year_built  floor_count
0   Education         7432      2008.0          NaN
1   Education         7432      2008.0          NaN
2   Education         7432      2008.0          NaN
3   Education         7432      2008.0          NaN
4   Education         7432      2008.0          NaN
```

Объединим получившийся набор с данными по погоде. Выставим индексы для объединения - timestamp, site_id.

Ввод [31]:
```
energy_0.set_index(["timestamp", "site_id"], inplace=True)
weather.set_index(["timestamp", "site_id"], inplace=True)
```

Проведем объединение и сбросим индексы.

Ввод [32]:
```
energy_0 = pd.merge(left=energy_0, right=weather, how="left",
                    left_index=True, right_index=True)
energy_0.reset_index(inplace=True)
print (energy_0.head())
```

```
             timestamp  site_id  building_id  meter  meter_reading  \
0  2016-01-01 00:00:00        0            0      0            0.0
1  2016-01-01 01:00:00        0            0      0            0.0
2  2016-01-01 02:00:00        0            0      0            0.0
3  2016-01-01 03:00:00        0            0      0            0.0
4  2016-01-01 04:00:00        0            0      0            0.0

  primary_use  square_feet  year_built  floor_count  air_temperature  \
0   Education         7432      2008.0          NaN             25.0
1   Education         7432      2008.0          NaN             24.4
2   Education         7432      2008.0          NaN             22.8
3   Education         7432      2008.0          NaN             21.1
4   Education         7432      2008.0          NaN             20.0

   cloud_coverage  dew_temperature  precip_depth_1_hr  sea_level_pressure  \
0             6.0             20.0                NaN              1019.7
1             NaN             21.1               -1.0              1020.2
2             2.0             21.1                0.0              1020.2
3             2.0             20.6                0.0              1020.1
4             2.0             20.0               -1.0              1020.0

   wind_direction  wind_speed
0             0.0         0.0
1            70.0         1.5
2             0.0         0.0
3             0.0         0.0
4           250.0         2.6
```

Нахождение пропущенных данных.

После объединения найдем пропущенные данные для дальнейшего заполнения. Посчитаем количество пропусков данных по столбцам:

Ввод [35]:
```
for column in energy_0.columns:
    energy_nulls = energy_0[column].isnull().sum()
    if energy_nulls > 0:
        print (column + ": " + str(energy_nulls))
print (energy_0[energy_0["precip_depth_1_hr"].isnull()])
```

```
floor_count: 8784
air_temperature: 3
cloud_coverage: 3830
dew_temperature: 3
precip_depth_1_hr: 1
sea_level_pressure: 85
wind_direction: 250
             timestamp  site_id  building_id  meter  meter_reading  \
0  2016-01-01 00:00:00        0            0      0            0.0

  primary_use  square_feet  year_built  floor_count  air_temperature  \
0   Education         7432      2008.0          NaN             25.0

   cloud_coverage  dew_temperature  precip_depth_1_hr  sea_level_pressure  \
0             6.0            20.0                NaN              1019.7

   wind_direction  wind_speed
0             0.0         0.0
```

Заполнение пропущенных данных уникально для каждой задачи. В данной оно будет организовано по следующему алгоритму:

- ✓ air_temperature: NaN → (заменяем на …) 0
- ✓ cloud_coverage: NaN → 0
- ✓ dew_temperature: NaN → 0
- ✓ precip_depth_1_hr: NaN → 0, −1 → 0
- ✓ sea_level_pressure: NaN → среднее
- ✓ wind_direction: NaN → среднее (роза ветров)

Ввод [42]:
```
energy_0["air_temperature"].fillna(0, inplace=True)
energy_0["cloud_coverage"].fillna(0, inplace=True)
energy_0["dew_temperature"].fillna(0, inplace=True)
energy_0["precip_depth_1_hr"] = energy_0["precip_depth_1_hr"].apply(lambda x:x if x>0 else 0)
energy_0_sea_level_pressure_mean = energy_0["sea_level_pressure"].mean()
energy_0["sea_level_pressure"] = energy_0["sea_level_pressure"].apply(lambda x:energy_0_sea_level_pressure_mean if x!=x
energy_0_wind_direction_mean = energy_0["wind_direction"].mean()
energy_0["wind_direction"] = energy_0["wind_direction"].apply(lambda x:energy_0_wind_direction_mean if x!=x else x)
energy_0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 16 columns):
timestamp             8784 non-null object
site_id               8784 non-null int64
building_id           8784 non-null int64
meter                 8784 non-null int64
meter_reading         8784 non-null float64
primary_use           8784 non-null object
square_feet           8784 non-null int64
year_built            8784 non-null float64
floor_count           0 non-null float64
air_temperature       8784 non-null float64
cloud_coverage        8784 non-null float64
dew_temperature       8784 non-null float64
precip_depth_1_hr     8784 non-null float64
sea_level_pressure    8784 non-null float64
wind_direction        8784 non-null float64
wind_speed            8784 non-null float64
dtypes: float64(10), int64(4), object(2)
memory usage: 1.1+ MB
```

ИНТЕРПОЛЯЦИЯ И ЭКСТРАПОЛЯЦИЯ

Постановка исходной задачи

Построить модель энергопотребления здания по часам. Погоду и характеристики здания пока не рассматривать.
Данные: http://video.ittensive.com/machine-learning/ashrae/train.0.0.csv.gz

Загрузка библиотек

В качестве первого шага логично и целесообразно загрузить библиотеки:

Ввод [22]:
```
%matplotlib inline
import numpy as np
import pandas as pd
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
from matplotlib.pyplot import rcParams
rcParams['figure.figsize'] = 16, 8
```

Загрузка данных

Затем переходим к загрузке непосредственно исходных данных:

Ввод [4]:
```
energy_0 = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.0.csv.gz")
print (energy_0.head())
```

```
   building_id  meter            timestamp  meter_reading
0            0      0  2016-01-01 00:00:00            0.0
1            0      0  2016-01-01 01:00:00            0.0
2            0      0  2016-01-01 02:00:00            0.0
3            0      0  2016-01-01 03:00:00            0.0
4            0      0  2016-01-01 04:00:00            0.0
```

Обогащение данных

Добавим серию с часом суток для построения суточной модели потребления:

Ввод [8]:
```
energy_0["timestamp"] = pd.to_datetime(energy_0["timestamp"])
energy_0["hour"] = energy_0["timestamp"].dt.hour
print (energy_0.head())
```

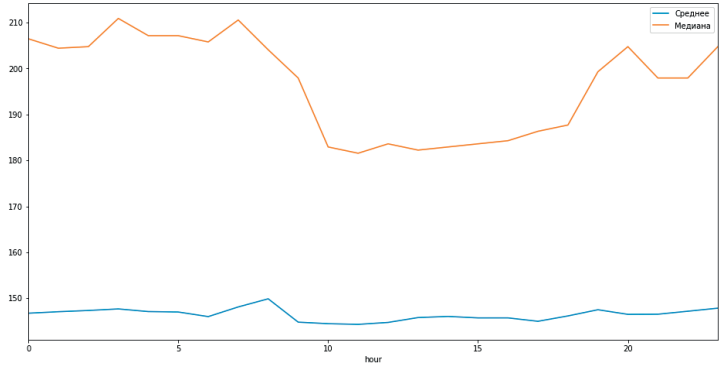
```
   building_id  meter            timestamp  meter_reading  hour
0            0      0  2016-01-01 00:00:00            0.0     0
1            0      0  2016-01-01 01:00:00            0.0     1
2            0      0  2016-01-01 02:00:00            0.0     2
3            0      0  2016-01-01 03:00:00            0.0     3
4            0      0  2016-01-01 04:00:00            0.0     4
```

Среднее потребление по часам

Выведем среднее и медиану потребления энергии по часам:

```
Ввод [19]: energy_0_hours = energy_0.groupby("hour")
           energy_0_averages = pd.DataFrame(
               {"Среднее": energy_0_hours.mean()["meter_reading"],
                "Медиана": energy_0_hours.median()["meter_reading"]})
           energy_0_averages.plot()
           plt.show()
```

На выходе работы ячейки будет получен следующий график:

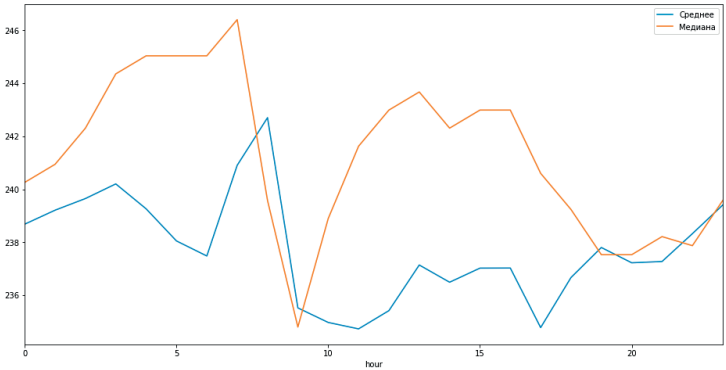


Фильтруем метрику

Удаляем нулевые значения из статистики:

```
Ввод [20]: energy_0_hours_filtered = energy_0[energy_0["meter_reading"]>0].groupby("hour")
           energy_0_averages_filtered = pd.DataFrame(
               {"Среднее": energy_0_hours_filtered.mean()["meter_reading"],
                "Медиана": energy_0_hours_filtered.median()["meter_reading"]})
           energy_0_averages_filtered.plot()
           plt.show()
```

Получая следующий результат функционирования:

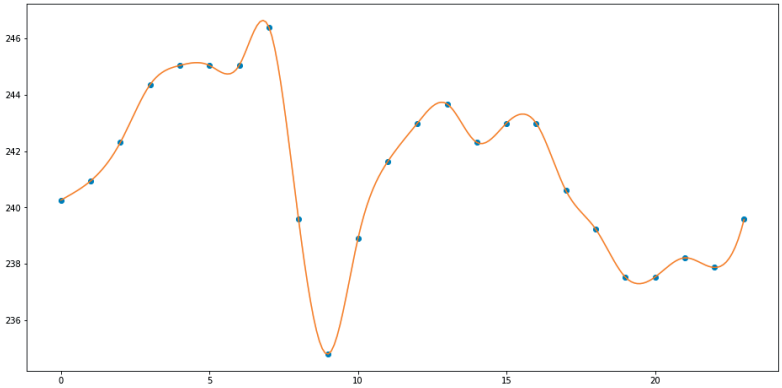


Интерполируем данные по часам

Построим модель внутрисуточного потребление энергии по зданию:

```
Ввод [30]: x = np.arange(0, 24)
           y = interp1d(x, energy_0_hours_filtered.median()["meter_reading"], kind="cubic")
           xn = np.arange(0, 23.1, 0.1)
           yn = y(xn)
           plt.plot(x, energy_0_hours_filtered.median()["meter_reading"],
                    'o', xn, yn, '-')
           plt.show()
```

Теперь график будет иметь такой вид:



ОЦЕНКА МОДЕЛИ

Постановка задачи

Построить простую модель энергопотребления здания по среднему значению, оценить эффективность модели через метрику

$$RMSLE = \sqrt{\frac{\sum_{i=1}^{n}\left(\log\left(p_i+1\right)-\log\left(a_i+1\right)\right)^2}{n}},$$

где n – число наблюдений;

log – натуральный логарифм;

p_i – вычисленное значение метрики;

a_i – заданное значение метрики.

Данные: http://video.ittensive.com/machine-learning/ashrae/train.0.0.csv.gz

Загрузка библиотек

Дополнительно сразу отсечем пустые дни и выделим час из значения времени:

Ввод [4]:
```
energy_0 = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.0.csv.gz")
energy_0 = energy_0[energy_0["meter_reading"] > 0]
energy_0["timestamp"] = pd.to_datetime(energy_0["timestamp"])
energy_0["hour"] = energy_0["timestamp"].dt.hour
print (energy_0.head())
```

```
      building_id  meter           timestamp  meter_reading  hour
704             0      0 2016-01-30 08:00:00        43.6839     8
725             0      0 2016-01-31 05:00:00        37.5408     5
737             0      0 2016-01-31 17:00:00        52.5571    17
2366            0      0 2016-04-08 14:00:00        59.3827    14
2923            0      0 2016-05-01 19:00:00       448.0000    19
```

Загрузка данных

Дополнительно сразу отсечем пустые дни и выделим час из значения времени:

Ввод [4]:
```
energy_0 = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.0.csv.gz")
energy_0 = energy_0[energy_0["meter_reading"] > 0]
energy_0["timestamp"] = pd.to_datetime(energy_0["timestamp"])
energy_0["hour"] = energy_0["timestamp"].dt.hour
print (energy_0.head())
```

```
      building_id  meter           timestamp  meter_reading  hour
704             0      0 2016-01-30 08:00:00        43.6839     8
725             0      0 2016-01-31 05:00:00        37.5408     5
737             0      0 2016-01-31 17:00:00        52.5571    17
2366            0      0 2016-04-08 14:00:00        59.3827    14
2923            0      0 2016-05-01 19:00:00       448.0000    19
```

Разделение данных на обучение и проверку

Выделим 20 % всех данных на проверку, остальные оставим на обучение:

Ввод [67]:
```
energy_0_train, energy_0_test = train_test_split(energy_0, test_size=0.2)
print (energy_0_train.head())
```

```
      building_id  meter           timestamp  meter_reading  hour
7193            0      0 2016-10-26 17:00:00        183.609    17
5842            0      0 2016-08-31 10:00:00        241.626    10
4081            0      0 2016-06-19 01:00:00        254.595     1
3997            0      0 2016-06-15 13:00:00        299.644    13
7140            0      0 2016-10-24 12:00:00        273.024    12
```

Создадим модели

Среднее и медианное значение потребление энергии по часам:

Ввод [68]:
```
energy_0_train_hours = energy_0_train.groupby("hour")
energy_0_train_averages = pd.DataFrame(
    {"Среднее": energy_0_train_hours.mean()["meter_reading"],
     "Медиана": energy_0_train_hours.median()["meter_reading"]})
print (energy_0_train_averages)
```

```
          Среднее   Медиана
hour
0      238.141292  240.2610
1      237.492207  240.2610
2      239.963320  242.3090
3      238.892307  244.3570
4      236.675818  244.3570
5      240.579716  245.7220
6      236.355284  242.3090
```

```
7     241.121046  247.0870
8     242.814911  239.2375
9     235.611241  235.4830
10    237.642876  239.5790
11    234.733561  241.6260
12    235.568616  242.3090
13    236.679616  244.3570
14    236.369121  242.9915
15    238.101027  243.3325
16    236.741289  242.9910
17    233.032625  239.5790
18    235.582897  239.5790
19    237.350518  237.5310
20    236.394606  236.8480
21    239.661546  238.8960
22    238.209773  237.5310
23    237.832567  239.2375
```

Функция проверки модели

$$RMSLE = \sqrt{\frac{\sum_{i=1}^{n}\left(\log\left(p_i+1\right)-\log\left(a_i+1\right)\right)^2}{n}}.$$

Для вычисления метрики создадим шесть новых столбцов в тестовом наборе данных: с логарифмом значения метрики, предсказанием по среднему и по медиане, а также с квадратом разницы предсказаний и логарифма значения. Последний столбец добавим, чтобы сравнить предсказание с его отсутствием – нулями в значениях:

Ввод [71]:
```
def calculate_model (x):
    meter_reading_log = np.log(x.meter_reading + 1)
    meter_reading_mean = np.log(energy_0_train_averages["Среднее"][x.hour] + 1)
    meter_reading_median = np.log(energy_0_train_averages["Медиана"][x.hour] + 1)
    x["meter_reading_mean_q"] = (meter_reading_log - meter_reading_mean)**2
    x["meter_reading_median_q"] = (meter_reading_log - meter_reading_median)**2
    x["meter_reading_zero_q"] = (meter_reading_log)**2
    return x

energy_0_test = energy_0_test.apply(calculate_model,
                                    axis=1, result_type="expand")
print (energy_0_test.head())
```

```
      building_id  meter           timestamp  meter_reading  hour  \
5238            0      0 2016-08-06 06:00:00        258.008     6
5460            0      0 2016-08-15 12:00:00        312.613    12
4714            0      0 2016-07-15 10:00:00        303.739    10
4022            0      0 2016-06-16 14:00:00        301.009    14
3511            0      0 2016-05-26 07:00:00        202.038     7

      meter_reading_log  meter_reading_mean  meter_reading_median  \
5238           5.556859            5.469558              5.494332
5460           5.748160            5.466238              5.494332
4714           5.719456            5.474968              5.483049
4022           5.710457            5.469616              5.497133
3511           5.313393            5.489438              5.513779

      meter_reading_mean_q  meter_reading_median_q  meter_reading_zero_q
5238              0.007621                0.003910             30.878681
5460              0.079480                0.064428             33.041340
4714              0.059774                0.055888             32.712173
4022              0.058004                0.045507             32.609317
3511              0.030992                0.040155             28.232147
```

Теперь остается просуммировать квадраты расхождений, разделить на количество значений и извлечь квадратный корень:

Ввод [72]:
```
energy_0_test_median_rmsle = np.sqrt(energy_0_test["meter_reading_median_q"].sum() / len(energy_0_test))
energy_0_test_mean_rmsle = np.sqrt(energy_0_test["meter_reading_mean_q"].sum() / len(energy_0_test))
energy_0_test_zero_rmsle = np.sqrt(energy_0_test["meter_reading_zero_q"].sum() / len(energy_0_test))
print ("Качество медианы:", energy_0_test_median_rmsle)
print ("Качество среднего:", energy_0_test_mean_rmsle)
print ("Качество нуля:", energy_0_test_zero_rmsle)
```

```
Качество медианы: 0.2541178189989124
Качество среднего: 0.2524486012205327
Качество нуля: 5.457644152335483
```

ЛИНЕЙНАЯ РЕГРЕССИЯ

Постановка задачи

Построить модель линейной регрессии энергопотребления здания, используя температуру воздуха (air_temperature) и влажность (dew_temperature).

Рассчитать качество построенной модели по проверочным данным.

Данные:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz
2. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/train.0.0.csv.gz

Подключение библиотек

Ввод [32]:
```
%matplotlib inline
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

Загрузка данных

Ввод [42]:
```
buildings = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz")
weather = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz")
energy_0 = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.0.csv.gz")
print (energy_0.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 4 columns):
building_id      8784 non-null int64
meter            8784 non-null int64
timestamp        8784 non-null object
meter_reading    8784 non-null float64
dtypes: float64(1), int64(2), object(1)
memory usage: 274.6+ KB
None
```

Объединение данных и фильтрация

Ввод [43]:
```
energy_0 = pd.merge(left=energy_0, right=buildings, how="left",
                    left_on="building_id", right_on="building_id")
energy_0.set_index(["timestamp", "site_id"], inplace=True)
weather.set_index(["timestamp", "site_id"], inplace=True)
energy_0 = pd.merge(left=energy_0, right=weather, how="left",
                    left_index=True, right_index=True)
energy_0.reset_index(inplace=True)
energy_0 = energy_0[energy_0["meter_reading"] > 0]
energy_0["timestamp"] = pd.to_datetime(energy_0["timestamp"])
energy_0["hour"] = energy_0["timestamp"].dt.hour
print (energy_0.head())
```

	timestamp	site_id	building_id	meter	meter_reading	\
704	2016-01-30 08:00:00	0	0	0	43.6839	
725	2016-01-31 05:00:00	0	0	0	37.5408	
737	2016-01-31 17:00:00	0	0	0	52.5571	
2366	2016-04-08 14:00:00	0	0	0	59.3827	
2923	2016-05-01 19:00:00	0	0	0	448.0000	

	primary_use	square_feet	year_built	floor_count	air_temperature	\
704	Education	7432	2008.0	NaN	8.3	
725	Education	7432	2008.0	NaN	12.8	
737	Education	7432	2008.0	NaN	20.6	
2366	Education	7432	2008.0	NaN	21.7	
2923	Education	7432	2008.0	NaN	31.1	

	cloud_coverage	dew_temperature	precip_depth_1_hr	sea_level_pressure	\
704	NaN	6.1	0.0	1019.0	
725	NaN	10.0	0.0	1021.9	
737	NaN	11.7	0.0	1020.9	
2366	2.0	14.4	0.0	1015.1	
2923	NaN	17.2	0.0	1016.1	

	wind_direction	wind_speed	hour
704	220.0	2.1	8
725	0.0	0.0	5
737	110.0	1.5	17
2366	250.0	3.1	14
2923	100.0	4.1	19

Добавление часа в данные

Ввод [9]:
```
energy_0["timestamp"] = pd.to_datetime(energy_0["timestamp"])
energy_0["hour"] = energy_0["timestamp"].dt.hour
```

Разделение данных на обучение и проверку

Ввод [44]:
```
energy_0_train, energy_0_test = train_test_split(energy_0, test_size=0.2)
print (energy_0_train.head())
```

	timestamp	site_id	building_id	meter	meter_reading	\
7066	2016-10-21 10:00:00	0	0	0	255.2780	
6762	2016-10-08 18:00:00	0	0	0	236.1660	
3473	2016-05-24 17:00:00	0	0	0	245.7220	
8211	2016-12-08 03:00:00	0	0	0	88.7328	
6259	2016-09-17 19:00:00	0	0	0	241.6260	

	primary_use	square_feet	year_built	floor_count	air_temperature	\
7066	Education	7432	2008.0	NaN	20.6	
6762	Education	7432	2008.0	NaN	30.6	
3473	Education	7432	2008.0	NaN	30.0	
8211	Education	7432	2008.0	NaN	18.9	
6259	Education	7432	2008.0	NaN	32.2	

```
      cloud_coverage  dew_temperature  precip_depth_1_hr  sea_level_pressure  \
7066             NaN             19.4                0.0              1010.2
6762             6.0             21.7                0.0              1008.7
3473             2.0             18.9                0.0              1019.4
8211             2.0             14.4                0.0              1019.6
6259             NaN             22.8               -1.0              1015.7

      wind_direction  wind_speed  hour
7066           310.0         2.6    10
6762           270.0         5.1    18
3473           100.0         4.1    17
8211           330.0         1.5     3
6259             0.0         0.0    19
```

Модель линейной регрессии и среднее

meter_reading = A * air_temperature + B * dew_temperature + C

Дополнительно вычислим среднее по часам, чтобы сравнить линейную регрессию с более простой моделью:

Ввод [45]:
```
energy_0_train_averages = energy_0_train.groupby("hour").mean()["meter_reading"]

energy_0_train_lr = pd.DataFrame(energy_0_train,
    columns=["meter_reading", "air_temperature", "dew_temperature"])
y = energy_0_train_lr["meter_reading"]
x = energy_0_train_lr.drop(labels=["meter_reading"], axis=1)
model = LinearRegression().fit(x, y)
print (model.coef_, model.intercept_)
```

```
[2.11796191 4.10628099] 104.74027151118469
```

Оценка модели

Ввод [46]:
```
def calculate_model (x):
    meter_reading_log = np.log(x.meter_reading + 1)
    meter_reading_mean = np.log(energy_0_train_averages[x.hour] + 1)
    meter_reading_lr = np.log(1 + x.air_temperature * model.coef_[0] +
                              x.dew_temperature * model.coef_[1] +
                              model.intercept_)
    x["meter_reading_lr_q"] = (meter_reading_log - meter_reading_lr)**2
    x["meter_reading_mean_q"] = (meter_reading_log - meter_reading_mean)**2
    return x

energy_0_test = energy_0_test.apply(calculate_model,
                                    axis=1, result_type="expand")
energy_0_test_lr_rmsle = np.sqrt(energy_0_test["meter_reading_lr_q"].sum() / len(energy_0_test))
energy_0_test_mean_rmsle = np.sqrt(energy_0_test["meter_reading_mean_q"].sum() / len(energy_0_test))
print ("Качество среднего:", energy_0_test_mean_rmsle)
print ("Качество линейной регрессии:", energy_0_test_lr_rmsle)
```

```
Качество среднего: 0.25581385523783384
Качество линейной регрессии: 0.21558045009847157
```

ОПТИМИЗАЦИЯ ПОТРЕБЛЕНИЯ ПАМЯТИ

Постановка задачи

Загрузить данные по энергопотреблению всех зданий в оперативную память и добиться ее минимального расхода

Данные:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz
2. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz

Подключение библиотек

Ввод [2]:
```
import pandas as pd
import numpy as np
```

Точность и размер типов

Ввод [5]:
```
for type_ in ["f2", "f4"]:
    print (np.finfo(type_))
for type_ in ["i1", "i2", "i4"]:
    print (np.iinfo(type_))
```

```
Machine parameters for float16
---------------------------------------------------------------
precision =   3   resolution = 1.00040e-03
machep =    -10   eps =        9.76562e-04
negep =     -11   epsneg =     4.88281e-04
minexp =    -14   tiny =       6.10352e-05
maxexp =     16   max =        6.55040e+04
nexp =        5   min =        -max
---------------------------------------------------------------

Machine parameters for float32
---------------------------------------------------------------
precision =   6   resolution = 1.0000000e-06
machep =    -23   eps =        1.1920929e-07
negep =     -24   epsneg =     5.9604645e-08
minexp =   -126   tiny =       1.1754944e-38
maxexp =    128   max =        3.4028235e+38
nexp =        8   min =        -max
---------------------------------------------------------------

Machine parameters for int8
---------------------------------------------------------------
min = -128
max = 127
---------------------------------------------------------------

Machine parameters for int16
---------------------------------------------------------------
min = -32768
max = 32767
---------------------------------------------------------------

Machine parameters for int32
---------------------------------------------------------------
min = -2147483648
max = 2147483647
---------------------------------------------------------------
```

Загрузка данных

Ввод [99]:
```
buildings = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz")
weather = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz")
energy = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz")
```

Потребление памяти

Ввод [100]:
```
print ("Строения: ", buildings.memory_usage().sum() / 1024**2, "Мб")
print ("Погода: ", weather.memory_usage().sum() / 1024**2, "Мб")
print ("Энергопотребление: ", energy.memory_usage().sum() / 1024**2, "Мб")
```

```
Строения:  0.06640625 Мб
Погода:  9.597526550292969 Мб
Энергопотребление:  368.06983947753906 Мб
```

Функция оптимизация памяти

Ввод [6]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        elif col == "timestamp":
            df[col] = pd.to_datetime(df[col])
        elif str(col_type)[:8] != "datetime":
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на',
          round(start_mem - end_mem, 2),
          'Мб (минус',
          round(100 * (start_mem - end_mem) / start_mem, 1),
         '%)')
    return df
```

Оптимизация памяти: строения

Ввод [102]:
```
buildings = reduce_mem_usage(buildings)
print (buildings.info())
```

```
Потребление памяти меньше на 0.05 Мб (минус 73.8 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1449 entries, 0 to 1448
Data columns (total 6 columns):
site_id        1449 non-null int8
building_id    1449 non-null int16
primary_use    1449 non-null category
square_feet    1449 non-null int32
year_built     675 non-null float16
floor_count    355 non-null float16
dtypes: category(1), float16(2), int16(1), int32(1), int8(1)
memory usage: 17.8 KB
None
```

Оптимизация: погода

Ввод [103]:
```
weather = reduce_mem_usage(weather)
print (weather.info())
```

```
Потребление памяти меньше на 6.53 Мб (минус 68.1 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 139773 entries, 0 to 139772
Data columns (total 9 columns):
site_id                139773 non-null int8
timestamp              139773 non-null datetime64[ns]
air_temperature        139718 non-null float16
cloud_coverage         70600 non-null float16
dew_temperature        139660 non-null float16
precip_depth_1_hr      89484 non-null float16
sea_level_pressure     129155 non-null float16
wind_direction         133505 non-null float16
wind_speed             139469 non-null float16
dtypes: datetime64[ns](1), float16(7), int8(1)
memory usage: 3.1 MB
None
```

Оптимизация: энергопотребление

Ввод [104]:
```
energy = reduce_mem_usage(energy)
print (energy.info())
```

```
Потребление памяти меньше на 195.54 Мб (минус 53.1 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12060910 entries, 0 to 12060909
Data columns (total 4 columns):
building_id      int16
meter            int8
timestamp        datetime64[ns]
meter_reading    float32
dtypes: datetime64[ns](1), float32(1), int16(1), int8(1)
memory usage: 172.5 MB
None
```

Объединение данных

Ввод [105]:
```
energy = pd.merge(left=energy, right=buildings, how="left",
                  left_on="building_id", right_on="building_id")
energy = pd.merge(left=energy.set_index(["timestamp", "site_id"]),
                  right=weather.set_index(["timestamp", "site_id"]),
                  how="left", left_index=True, right_index=True)
energy.reset_index(inplace=True)
energy = energy.drop(columns=["site_id", "meter"], axis=1)
print (energy.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12060910 entries, 0 to 12060909
Data columns (total 14 columns):
timestamp              datetime64[ns]
building_id            int16
meter_reading          float32
primary_use            category
square_feet            int32
year_built             float16
floor_count            float16
air_temperature        float16
cloud_coverage         float16
dew_temperature        float16
precip_depth_1_hr      float16
sea_level_pressure     float16
wind_direction         float16
wind_speed             float16
dtypes: category(1), datetime64[ns](1), float16(9), float32(1), int16(1), int32(1)
memory usage: 425.6 MB
None
```

Исследование диапазона данных

Ввод [106]:
```
print ("wind speed:", sorted(energy["wind_speed"].unique()))
print ("cloud coverage:", sorted(energy["cloud_coverage"].unique()))
print ("precip depth:", sorted(energy["precip_depth_1_hr"].unique()))
```

```
wind speed: [ 0.          3.09960938  3.59960938  4.1015625          nan  1.5
  5.          2.59960938  6.19921875  1.          3.          4.6015625
  0.5         2.09960938  7.69921875  4.          5.1015625   5.69921875
  7.19921875  8.203125    9.296875   11.          6.69921875  8.796875
 12.          9.796875   14.         13.         12.8984375  10.796875
 10.296875   11.296875   10.          9.          7.          6.
  2.          6.1015625   8.          1.59960938  1.29980469  3.19921875
 13.3984375  11.796875   12.3984375  12.796875    8.703125   13.8984375
 15.         16.         17.         18.         19.         14.8984375
 14.3984375  15.3984375  16.5        18.5         2.19921875]
cloud coverage: [ 6. nan  8.  4.  0.  2.  7.  5.  3.  9.  1.]
precip depth: [ nan  -1.   0.   5.   2.   3.   8.  13.  10.   7.  11.  15.  18.  20.
  48.  25.  23.  41.  28.  66.  36.  33.  38.  46.  30.  17.  76.  53.
  94.  58.   9.   4.   6. 165.  51.  61.  81.  69.  56.  79. 112.  86.
  64.  89.  21.  43.  39.  97.  24.  26.  84.  16.  91.  45.  40.  34.
  14.  12. 203.  71.  22.  27. 122. 343. 130. 135. 124.  68.  37.  19.
 193. 155. 102. 109.  74. 198. 310. 160. 234.  35.  32. 105. 103. 104.
 196.  29.  99. 114. 119.  31.  50. 142.  55. 150. 262.  70. 137. 340.
 127. 145. 163. 132. 333. 175.  83. 201. 257. 162. 164. 107.  98.  47.
 152.  42. 216. 241. 221. 113. 147. 239. 217. 211. 140.  73.  78. 236.
 180.  60. 191.]
```

Приведение к целым типам

Ввод [2]:
```
def round_fillna (df, columns):
    for col in columns:
        type_ = "int8"
        if col in ["wind_direction", "year_built", "precip_depth_1_hr"]:
            type_ = "int16"
        if col == "precip_depth_1_hr":
            df[col] = df[col].apply(lambda x:0 if x<0 else x)
        df[col] = np.round(df[col].fillna(value=0)).astype(type_)
    return df
```

EDA И ИССЛЕДОВАНИЕ ЗАВИСИМОСТЕЙ В ДАННЫХ

Постановка задачи

EDA – Exploratory Data Analysis

Проверить взаимосвязи между параметрами измерений, найти потенциальные особенности для построения модели

Данные:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz
2. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz

Подключение библиотек

Ввод [11]:
```
%matplotlib inline
import pandas as pd
import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
from matplotlib.pyplot import rcParams
import seaborn as sns
rcParams['figure.figsize'] = 16, 8
```

Загрузка данных

Ввод [2]:
```
buildings = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz")
weather = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz")
energy = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz")
```

Оптимизация памяти

Ввод [3]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        elif col == "timestamp":
            df[col] = pd.to_datetime(df[col])
        elif str(col_type)[:8] != "datetime":
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
    return df

def round_fillna (df, columns):
    for col in columns:
        type_ = "int8"
        if col in ["wind_direction", "year_built", "precip_depth_1_hr"]:
            type_ = "int16"
        if col == "precip_depth_1_hr":
            df[col] = df[col].apply(lambda x:0 if x<0 else x)
        df[col] = np.round(df[col].fillna(value=0)).astype(type_)
    return df
```

Ввод [4]:
```
buildings = reduce_mem_usage(buildings)
weather = reduce_mem_usage(weather)
energy = reduce_mem_usage(energy)
```

```
Потребление памяти меньше на 0.05 Мб (минус 73.8 %)
Потребление памяти меньше на 6.53 Мб (минус 68.1 %)
Потребление памяти меньше на 195.54 Мб (минус 53.1 %)
```

Объединение данных и очистка

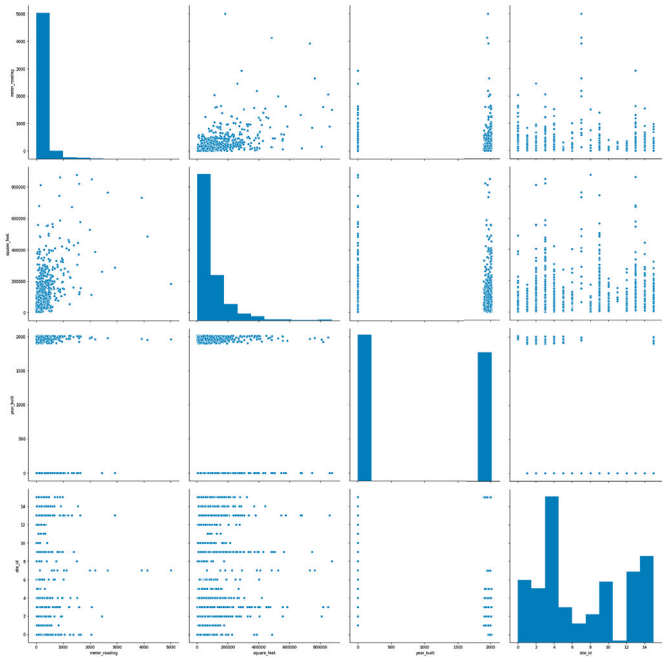
Ввод [5]:
```
energy = pd.merge(left=energy, right=buildings, how="left",
                  left_on="building_id", right_on="building_id")
energy = energy.set_index(["timestamp", "site_id"])
weather = weather.set_index(["timestamp", "site_id"])
energy = pd.merge(left=energy, right=weather, how="left",
                  left_index=True, right_index=True)
```

```
energy.reset_index(inplace=True)
energy = energy.drop(columns=["meter"], axis=1)
energy = round_fillna(energy, ["wind_direction", "wind_speed",
        "cloud_coverage", "precip_depth_1_hr",
        "year_built", "floor_count"])
energy = energy[energy["meter_reading"] > 0]
del buildings
del weather
print (energy.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11530741 entries, 45 to 12060909
Data columns (total 15 columns):
timestamp             datetime64[ns]
site_id               int64
building_id           int16
meter_reading         float32
primary_use           category
square_feet           int32
year_built            int16
floor_count           int8
air_temperature       float16
cloud_coverage        int8
dew_temperature       float16
precip_depth_1_hr     int16
sea_level_pressure    float16
wind_direction        int16
wind_speed            int8
dtypes: category(1), datetime64[ns](1), float16(3), float32(1), int16(4), int32(1), int64(1), int8(3)
memory usage: 549.8 MB
None
```

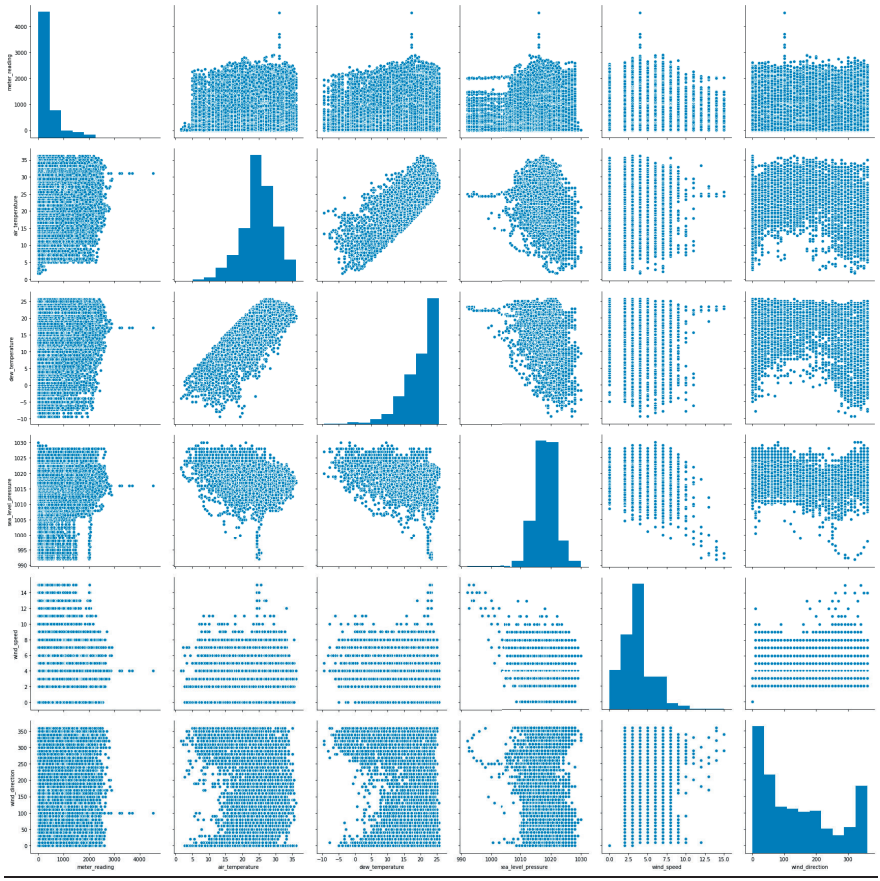
Поиск зависимостей: здания

Ввод [22]:
```
data_corr_meta = pd.DataFrame(energy.groupby("building_id").median(),
        columns=["meter_reading", "square_feet", "year_built", "site_id"])
data_corr_meta.dropna(inplace=True)
sns.pairplot(data_corr_meta, height=6)
plt.show()
del data_corr_meta
```



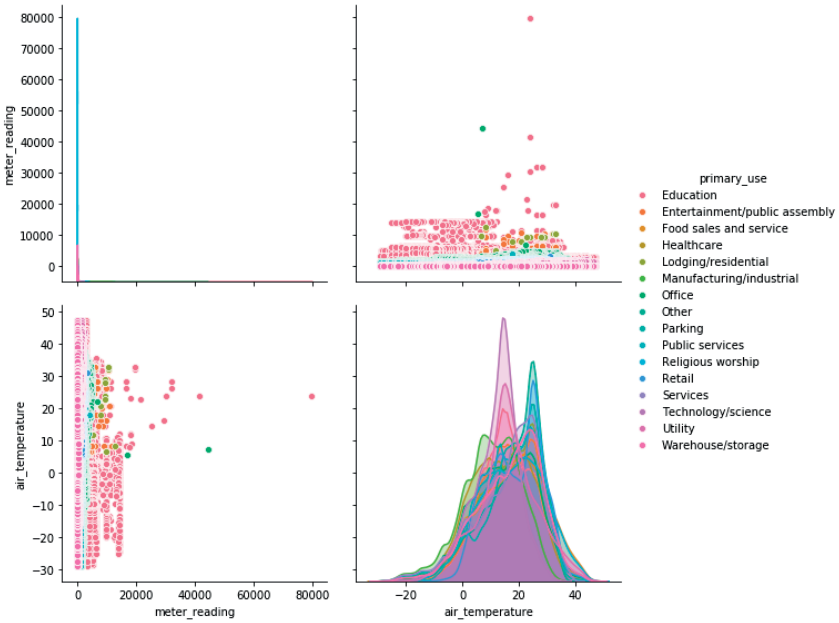
Поиск зависимостей: погода

Ввод [7]:
```
data_corr_weather = pd.DataFrame(energy[energy["site_id"] == 0],
    columns=["meter_reading", "air_temperature", "dew_temperature", "sea_level_pressure","wind_speed","wind_direction"])
data_corr_weather.dropna(inplace=True)
sns.pairplot(data_corr_weather, height=4)
plt.show()
del data_corr_weather
```



Поиск зависимостей: тип здания

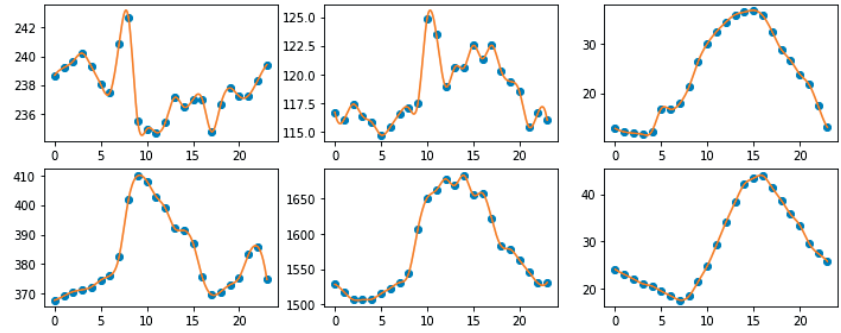
Ввод [8]:
```
data_corr_temp_primary = pd.DataFrame(energy,
    columns=["meter_reading", "air_temperature", "primary_use"])
data_corr_temp_primary.dropna(inplace=True)
sns.pairplot(data_corr_temp_primary, hue="primary_use", height=4)
plt.show()
del data_corr_temp_primary
```

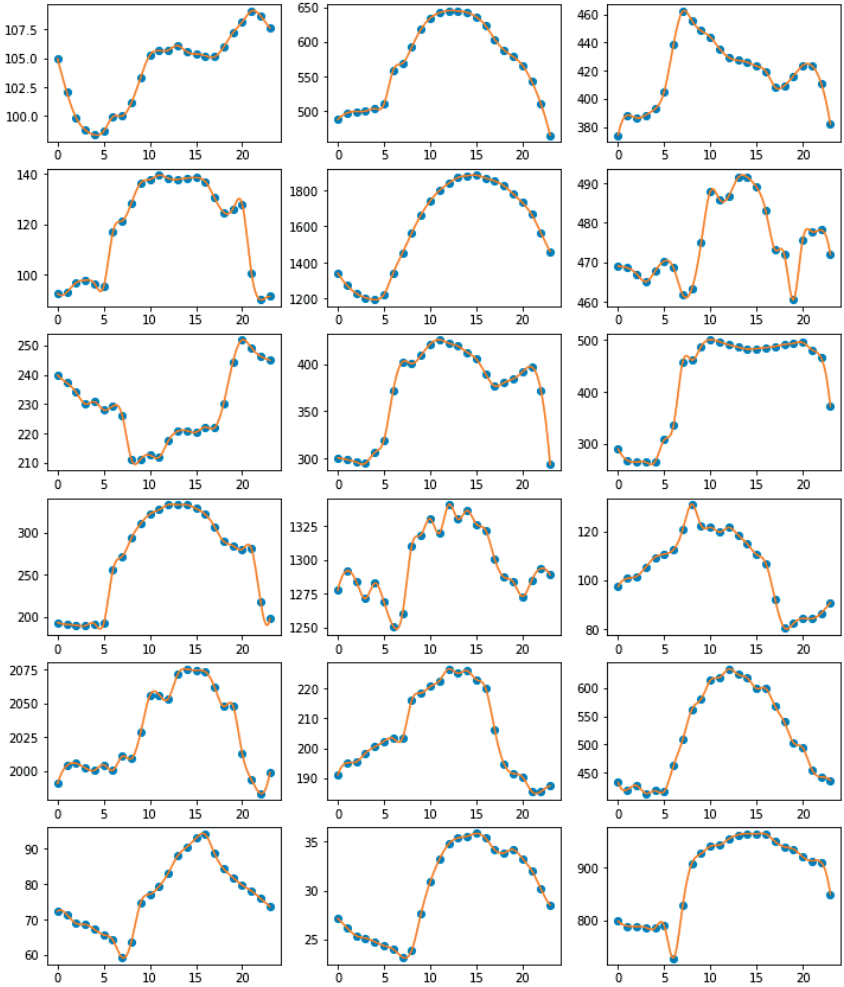


Поиск зависимостей: время

Ввод [12]:

```
energy["hour"] = energy["timestamp"].dt.hour.astype("int8")
fig = plt.figure(figsize=(12,20))
for i in range(0, 24):
    fig.add_subplot(8, 3, i+1)
    train_df_i = energy[(energy["building_id"]==i) & (energy["meter_reading"]>0)]
    train_df_i_hourly = train_df_i.groupby("hour").mean().reset_index()
    x = np.arange(0, 24)
    y = interp1d(x, train_df_i_hourly["meter_reading"], kind="cubic")
    xn = np.arange(0, 23.1, 0.1)
    yn = y(xn)
    plt.plot(np.arange(0, 24),
             train_df_i_hourly["meter_reading"], 'o', xn, yn, '-')
plt.show()
```

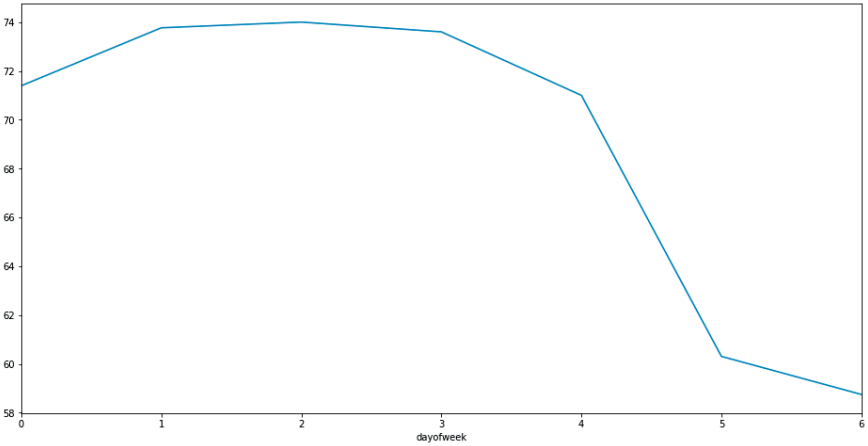


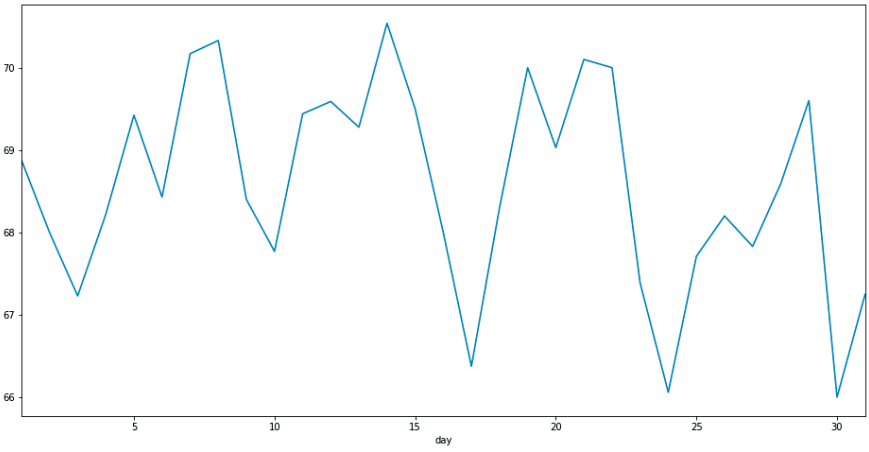


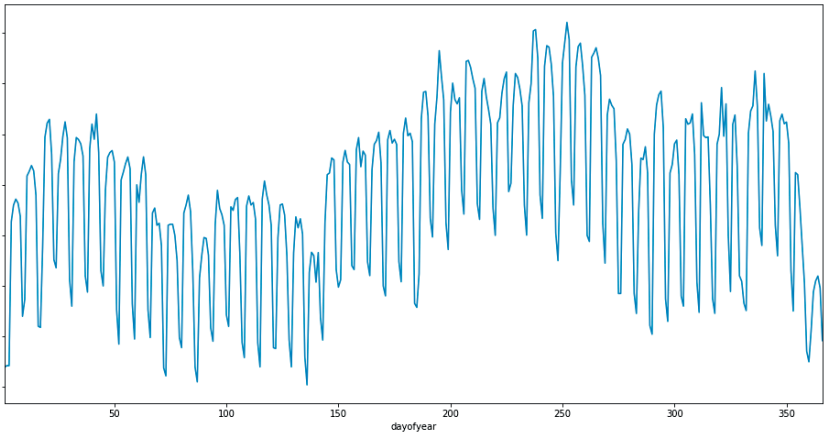
Поиск зависимостей: дата

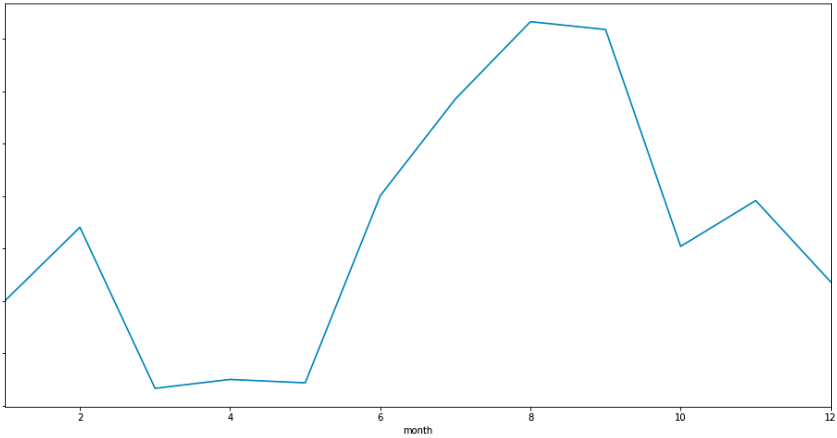
Ввод [18]:
```
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar

dates_range = pd.date_range(start='2015-12-31', end='2017-01-01')
us_holidays = calendar().holidays(start=dates_range.min(),
                                  end=dates_range.max())

energy["dayofweek"] = energy["timestamp"].dt.dayofweek.astype("int8")
energy["day"] = energy["timestamp"].dt.day.astype("int8")
energy["dayofyear"] = energy["timestamp"].dt.dayofyear.astype("int16")
energy["month"] = energy["timestamp"].dt.month.astype("int8")
energy["week"] = energy["timestamp"].dt.week.astype("int8")
energy["date"] = pd.to_datetime(energy["timestamp"].dt.date)
energy['is_holiday'] = (energy['date'].isin(us_holidays)).astype("int8")
```

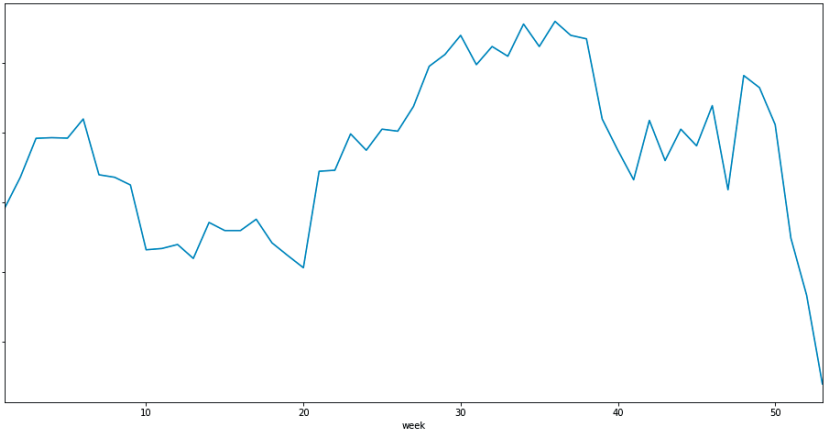
Ввод [19]:
```
energy.groupby("dayofweek").median()["meter_reading"].plot()
plt.show()
energy.groupby("day").median()["meter_reading"].plot()
plt.show()
energy.groupby("dayofyear").median()["meter_reading"].plot()
plt.show()
energy.groupby("month").median()["meter_reading"].plot()
plt.show()
energy.groupby("week").median()["meter_reading"].plot()
plt.show()
energy.groupby("is_holiday").median()["meter_reading"].plot()
plt.show()
```

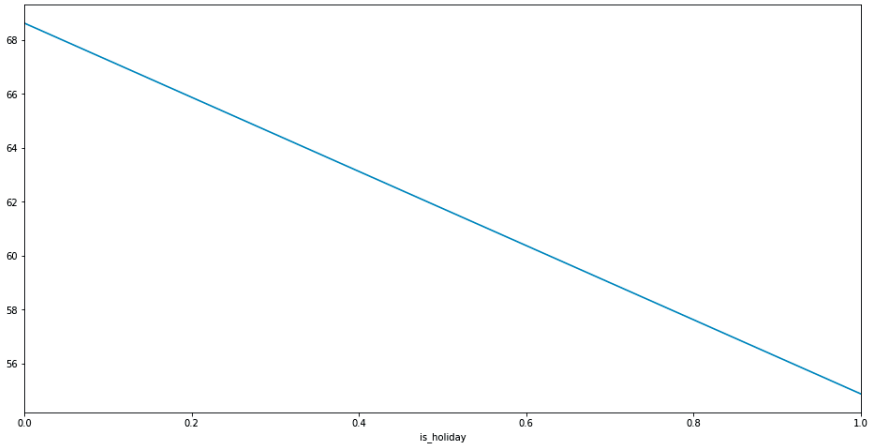





85
80
75
70
65
60
55
50
50
100
150
200
250
300
350
dayofyear


76
74
72
70
68
66
64
62
2
4
6
8
10
12
month


75
70
65
60
55
10
20
30
40
50
week



Очистка памяти

```
Ввод [ ]: del energy
```

ЗАПОЛНЕНИЕ ПРОПУСКОВ В ДАННЫХ

Постановка задачи

Заполним отсутствующие значения по погоде интерполяционными данными.

Посчитаем модель линейной регрессии по первому зданию и найдем ее точность.

Данные:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz
2. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz

Подключение библиотек

```
Ввод [32]: import pandas as pd
           import numpy as np
           from scipy.interpolate import interp1d
           from sklearn.model_selection import train_test_split
           from sklearn.linear_model import LinearRegression
```

Загрузка данных

Ввод [33]:
```
buildings = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz")
weather = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz")
energy = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz")
```

Отсечение здания 0 и отсутствующих значений

Ввод [34]:
```
energy = energy[(energy["building_id"]==0)]
```

Объединение данных

Ввод [35]:
```
energy = pd.merge(left=energy, right=buildings, how="left",
                  left_on="building_id", right_on="building_id")
energy = energy.set_index(["timestamp", "site_id"])
weather = weather.set_index(["timestamp", "site_id"])
energy = pd.merge(left=energy, right=weather, how="left",
                  left_index=True, right_index=True)
energy.reset_index(inplace=True)
energy = energy.drop(columns=["meter", "site_id", "floor_count"], axis=1)
del buildings
del weather
print (energy.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 14 columns):
timestamp             8784 non-null object
building_id           8784 non-null int64
meter_reading         8784 non-null float64
primary_use           8784 non-null object
square_feet           8784 non-null int64
year_built            8784 non-null float64
floor_count           0 non-null float64
air_temperature       8781 non-null float64
cloud_coverage        4954 non-null float64
dew_temperature       8781 non-null float64
precip_depth_1_hr     8783 non-null float64
sea_level_pressure    8699 non-null float64
wind_direction        8534 non-null float64
wind_speed            8784 non-null float64
dtypes: float64(10), int64(2), object(2)
memory usage: 960.8+ KB
None
```

Оптимизация памяти

Ввод [1]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        elif col == "timestamp":
            df[col] = pd.to_datetime(df[col])
        elif str(col_type)[:8] != "datetime":
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 *
                                                                    (start_mem - end_mem) / start_mem, 1), '%)')
    return df
```

Ввод [37]:
```
energy = reduce_mem_usage(energy)
```

```
Потребление памяти меньше на 0.62 Mб (минус 66.1 %)
```

```
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:8: RuntimeWarning: in
valid value encountered in less

c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:10: RuntimeWarning: i
nvalid value encountered in less
  # Remove the CWD from sys.path while we load stuff.
```

Интерполяция данных

Ввод [38]:
```
energy["precip_depth_1_hr"] = energy["precip_depth_1_hr"].apply(lambda x:x if x>0 else 0)
interpolate_columns = ["air_temperature", "dew_temperature",
                       "cloud_coverage", "wind_speed",
                       "precip_depth_1_hr", "sea_level_pressure"]
for col in interpolate_columns:
    energy[col] = energy[col].interpolate(limit_direction='both',
                          kind='cubic')
```

Проверка качества интерполяции

Ввод [39]:
```
pd.set_option('use_inf_as_na', True)
for col in interpolate_columns:
    print (col, "Inf+NaN:", energy[col].isnull().sum())
```

```
air_temperature Inf+NaN: 0
dew_temperature Inf+NaN: 0
cloud_coverage Inf+NaN: 0
wind_speed Inf+NaN: 0
precip_depth_1_hr Inf+NaN: 0
sea_level_pressure Inf+NaN: 0
```

Разделение данных

Ввод [40]:
```
energy_train, energy_test = train_test_split(energy[energy["meter_reading"]>0], test_size=0.2)
print (energy_train.head())
```

```
                timestamp  building_id  meter_reading primary_use  square_feet  \
4265 2016-06-26 17:00:00            0        232.125   Education         7432
5267 2016-08-07 11:00:00            0        293.500   Education         7432
4255 2016-06-26 07:00:00            0        242.250   Education         7432
7387 2016-11-03 19:00:00            0        180.250   Education         7432
4560 2016-07-09 00:00:00            0        288.000   Education         7432

      year_built  floor_count  air_temperature  cloud_coverage  \
4265      2008.0          NaN        27.796875        4.000000
5267      2008.0          NaN        25.000000        6.332031
4255      2008.0          NaN        26.093750        5.000000
7387      2008.0          NaN        28.906250        5.000000
4560      2008.0          NaN        32.187500        6.000000

      dew_temperature  precip_depth_1_hr  sea_level_pressure  wind_direction  \
4265        22.203125                0.0              1020.0           150.0
5267        23.296875                0.0              1014.5           180.0
4255        22.796875                0.0              1018.5           170.0
7387        15.000000                0.0              1019.5            60.0
4560        22.203125                0.0              1017.0           300.0

      wind_speed
4265    3.099609
5267    3.099609
4255    2.599609
7387    5.101562
4560    4.601562
```

Линейная регрессия

Ввод [41]:
```
regression_columns = ["meter_reading", "air_temperature",
        "dew_temperature", "cloud_coverage", "wind_speed",
        "precip_depth_1_hr", "sea_level_pressure"]

energy_train_lr = pd.DataFrame(energy_train,
    columns=regression_columns)
y = energy_train_lr["meter_reading"]
x = energy_train_lr.drop(labels=["meter_reading"], axis=1)
model = LinearRegression().fit(x, y)
print (model.coef_, model.intercept_)
```

```
[ 2.57940166  3.66685087 -2.36935224 -1.97273319  0.13562709 -0.97385944] 1109.167249161545
```

Предсказание и оценка модели

Ввод [45]:
```
def calculate_model (x):
    lr = np.sum([x[col] * model.coef_[i] for i,col in enumerate(regression_columns[1:])])
    lr += model.intercept_
    x["meter_reading_lr_q"] = (np.log(1 + x.meter_reading) -
                               np.log(1 + lr))**2
    return x

energy_test = energy_test.apply(calculate_model,
                                axis=1, result_type="expand")
energy_test_lr_rmsle = np.sqrt(energy_test["meter_reading_lr_q"].sum() / len(energy_test))
print ("Качество линейной регрессии:", energy_test_lr_rmsle, round(energy_test_lr_rmsle, 1))
```

```
Качество линейной регрессии: 0.19696713505628877 0.2
```

Часть 3
МОДЕЛИ ЛИНЕЙНОЙ РЕГРЕССИИ

ЛИНЕЙНАЯ РЕГРЕССИЯ И L1_L2-РЕГУЛЯРИЗАЦИЯ

Рассмотрим уравнение линейной регрессии и оптимизации гиперпараметров модели линейной регрессии. Линейная регрессия говорит, что у есть некоторая линейная взаимосвязь между данными или предсказуемым значением и зависимые переменные или тем значением, которое мы хотим предсказать или вычислить. Например, имеется уравнение такого вида:

$$y = a_0 + a_1 x_1 + ... + a_n x_n.$$

Где x_1-x_n это значение данных, по которым предсказываем значение y то есть зависимой переменной, а $a_n x_n$ это коэффициенты перед этими данными, то есть они линейны, поскольку вычисляются по методу наименьших квадратов за счет минимизации ошибки, a_0 это свободный член показывает предсказуемые значения, в том случае если у все данные равны нулю и он содержит в себе первый шаг к оптимизации гиперпараметров.

Если считаем, что у нас предсказуемое значение y также должно равняться нулю, когда все данные равны нулю, то есть линия линейной регрессии выходит из ноля. В этом случае можно попробовать a_0 приравнять к нулю. То есть задать модель линейной регрессии таким образом что свободный член будет отсутствовать, получается, что вся модель будет определяться исключительно данными. В этом случае можно получить немного больше предсказательной точности.

Оптимизация гиперпараметров нужна для того, чтобы мы могли предсказать мир за пределами наших данных. Поскольку наши данные уже известны и именно под них подстраиваем модель, которая эти данные описывает. Однако, как только начинаем предсказывать, то, что за пределами известных данных, тут же получаем некоторую ошибку, и появляется задача на основании всех этих известных данных каким-то образом устранить эту предсказательную ошибку.

Она в том числе устраняется за счет того, что мы пытаемся немножко ухудшить модель, то есть чтобы она меньше подгонялась под данные, но при этом у нее увеличилась обобщающая способность и предсказательная способность и она соответственно лучше предсказывала неизвестные ей данные, по которым она не обучалась. Также это известно как некоторое смещение модели в ту сторону, которая нам кажется наиболее правильной.

Машинное обучение, и машина, в частности, не знает, что с этими данными делать, где правильная сторона. Тем не менее мы можем на основании

вычислений ошибки предсказаний, можем сдвинуть предсказание в какую-то сторону, которая нам кажется по нашему экспертному мнению, по тем данным что мы знаем.

Например, есть линейная зависимость или квадратичная зависимость или данные должны выходить строго из нуля. Мы обладая этим знанием пытаемся его наложить на модель и возможно немного ухудшить ее предсказать способность относительно текущих данных, но сильно улучшить ее относительно всех неизвестных данных, то есть сделать так чтобы все таки модель работала и она обобщала имеющиеся ситуации.

Первый шаг к обобщению – это устранение коэффициента свободного члена, таким образом модель может начать предсказывать ситуацию лучше.

Следующий шаг заключается в том, что к минимизации предсказательной ошибки нашей модели, которая выглядит как:

$$\sum_{i=1}^{n}(y - y_i)^2.$$

Для всех наших данных считаем ошибку предсказания, которую минимизируем. Первый подход: к этой ошибке добавим некоторое ограничение на веса линейной регрессии, но предполагая, что данные нормированы и что веса не должны быть слишком большими, они не должны друг от друга сильно отличаться. Предполагаем, что переменная вносит примерно равный вклад в модель и ограничения на веса на самом деле способствует более простой модели. Поскольку большие веса, то есть большие по абсолютному значению веса расшатывают модель.

Например, a_1 может быть очень большим положительным a_2 у будет очень большим отрицательным и метод наименьших квадратов выдаст те значения, которые будут минимизировать ошибку. Только мы знаем, что, если данные чуть изменится всю модель перекосит. И чтобы свести перекос к минимуму нужно ввести некоторые ограничения на веса a_1 a_n для того, чтобы они были не очень большими, чтобы при больших вариациях данных модель вела себя более стабильно.

Для этого вводят дополнительный член.

$$\sum_{i=1}^{N}(y - y_i)^2 + \lambda_1 \sum_{i=1}^{n}|a_i| \to \min.$$

Это называется λ_1 регуляризацией, либо лассо регуляризации когда нужно чтобы это выражение стремилось к минимуму. А именно к самой ошибке модели добавляем регуляризационный параметр, который минимизирует веса этой модели. И не нужно увеличивать веса даже если это уменьшит ошибку. Считаем, что предсказательная способность модели зависит от суммы модулей весов в том числе через параметр лямбда.

Такой вид оптимизации гиперпараметров для модели линейной регрессии называется λ_1 регуляризацией. В этой формуле кроме модуля может еще стоять квадрат.

$$\sum_{i=1}^{N}\left(y-y_i\right)^2+\lambda_2\sum_{i=1}^{n}\left(a_i\right)^2\to\min.$$

Эта регуляризация называется λ_2 регуляризацией или гребневой регуляризацией. Ее суть заключается что к матрице ошибок добавляем гребень, который увеличивает ошибку при попытке уйти в сторону, то есть если веса становится больше, а в формуле уже квадрат весов, то большие веса просто не могут получиться поскольку ошибка тут же начинает возрастать даже если в формуле идет уменьшение. То есть если начинаем предсказывать данные точнее, но с большими весами то раскачивания весов происходит быстрее и здесь веса ещё больше зажаты, потому что они зажаты через квадрат.

Для этого подхода нужно найти значение λ_1, λ_2. Для того чтобы их найти применяют подход эластичной сети ElastikNet. Суть заключается не в минимизации λ_1 или λ_2 а в минимизации их комбинации через следующую сумму:

$$\sum_{i=1}^{N}\left(y-y_i\right)^2+\lambda_1\sum_{i=1}^{n}\left|a_i\right|+\lambda_2\sum_{i=1}^{n}\left(a_i\right)^2\to\min.$$

В формуле уже два гиперпараметра которым нужно подобрать оптимальные значения, которые приведут к некоторым другим наборам наших параметров. Тем не менее этот другой набор будет обладать большей обобщающей силой. Но проверить это можно только по конечной метрики модели, если метрика уменьшается значит модель начинает работать лучше. Для этого используется некоторый проверочный набор данных, которые используются только для проверки качества модели.

Чтобы оптимизировать гиперпараметры, нужно сделать сетку. Поскольку в нашем случае λ_1, λ_2 могут быть примерно произвольными, то, например, выбираем сетку:

	λ1				
λ2	0,01	0,1	1	10	100
	0,1				
	1				
	10				
	100				

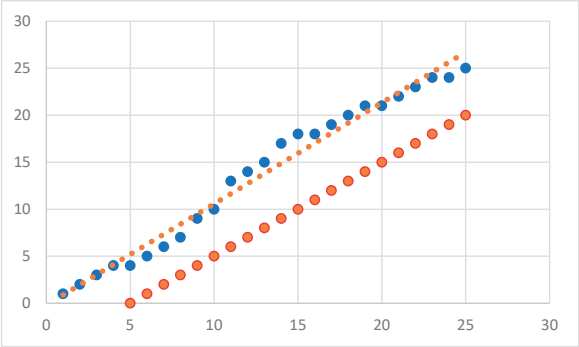
Получается сетка для оптимизации гиперпараметров модели линейной регрессии по методу эластичной сети. Нужно пройти жадным поиском и найти оптимальное значение сетки. Скажем что оптимальное значение будет в *.

	λ1				
λ2	0,01	0,1	1	10	100
	0,1		*		
	1				
	10				
	100				

И затем в окрестностях этой точки находим те значения гиперпараметров, которые будут оптимальным значением для обучающих данных. Далее на

тестовой выборке проверяем насколько эти гиперпараметры улучшают модель. Если на обучающей выборке гиперпараметры оптимизированы, то и на проверочной выборке идёт уменьшение ошибки, то есть используется не самое оптимальное решение, найденное по методу наименьших квадратов, а используется некоторые смещённое решение которые лучшим образом описывает исходные данные. Поскольку из-за статистической ошибки, мы не знаем в какую сторону смещены данные. Они могут быть смещены в любую сторону. Нужно предсказать обратное смещение данных.

Поскольку модель линейной регрессии отлично описывает смещенные данные, и теперь надо за счёт оптимизации параметров эту модель свести сторону и описать параметры более точно в плане их генеральной совокупности, то есть повысить обобщающую силу модели. Если взять некоторый набор данных:



Строим модель линейной регрессии *синие точки*. Однако, предполагаем, что истинная модель линейной регрессии находится на *красных точках*. Из-за того, что выборка смещена на некоторую константу и за счёт оптимизации гиперпараметров λ_1, λ_2 регуляризации пытаемся сместить линию обратно к идеальной.

При этом модель будет предсказывать исходный набор данных хуже, однако на других выборках из генеральной совокупности она будет работать лучше. В этом состоит основная идея оптимизации гиперпараметров модели линейной регрессии, из-за того, что можем изменять свободный член посмотреть на его влияние и изменять параметры регуляризации, чтобы добиться за счет смещения модели улучшения её обобщающей способности и улучшения её работы на выбранной нами метрике.

ИЗОТОНИЧЕСКАЯ РЕГРЕССИЯ

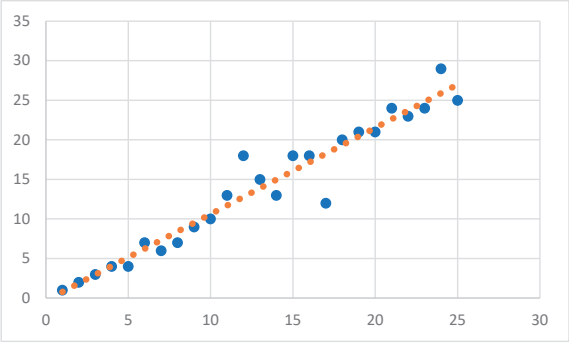
В ряде задач машинного обучения особенно при работе с малыми данными, когда требуется решить задачу регрессии, то есть определить некоторые

значения целевой переменной (ни класс, ни тренд, ни количество классов, ни число векторов, а именно некоторое число), которое наилучшим образом определяет новое неизвестное состояние системы.

Эта задача решается большим количеством методов в машинном обучении, включая линейную регрессию, реализуемую регрессию. Однако одним из подходов, которые может помочь в решении такой задачи является изотоническая регрессия.

В чём заключается изотоническая регрессия, рассмотрим на примере двух переменных. В отличие от линейной регрессии у изотонической регрессии строгие условия применения, в дополнения к условиям линейной регрессии данные должны быть монотонные, то есть данные должны монотонно возрастать, то есть предсказанное значение должно монотонно увеличивается, либо монотонно уменьшаться, то есть предсказанное значение должно монотонно уменьшаться. Важным условием применения регрессии для набора данных является монотонность этого набора.

Например, возьмем задачу на предсказывание положительности жизни людей по годам по этому графику:

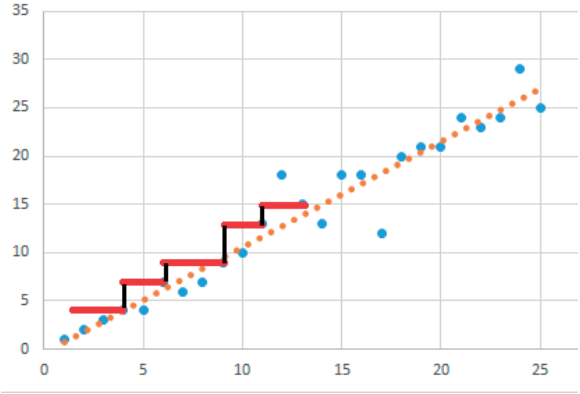


Продолжительность жизни располагается по оси X. Уравнение линии тренда вида $y = kx + b$. На графике видно, что часть точек имеет очень большой разброс, соответственно и большую ошибку. Изотоническая регрессия помогает уменьшить этот разброс. Изотоническую регрессию имеет смысл использовать в ансамбле с линейной регрессией.

Изотоническая регрессия представляет собой фиксируемое предсказуемое значение от независимых параметров.

Благодаря этому выстраивается изотонический график, получается, что эти предсказываемые значения будут иметь меньшую ошибку предсказания чем у линейной регрессии. Линейная регрессия является наилучшей прямой которая минимизирует ошибку. Изотоническое линия не является гладкой кривой, но, однако, она позволяет получить некоторые значения на отрезке значительно ближе, чем методом линейной регрессии. После усреднения значений этих двух регрессий, то есть суммирование значений, разделенное на

два, получим уже более точные предсказания чем просто с помощью линейной регрессии.



Пример использования нужно предсказать значение продолжительности жизни через пять лет. Наиболее точный ответ может дать ансамблирование различных методов регрессий. Одной из лучших моделей в случае явной монотонности зависимости целевой переменной от независимых параметров является именно эта комбинация линейной и изотонической регрессий.

BIC и AIC

В целом при выборе оптимальной модели машинного обучения и линейной регрессии в частности нужно ориентироваться на уменьшение ошибки, то есть увлечения предсказательной силы или обобщающей силы если можно её каким-то образом измерить. Тем не менее в ряде случаев это не всегда является хорошим критерием, то есть ошибка может уменьшаться. Однако модель будет усложняться, то есть нужно найти оптимальную точку баланса.

Этот баланс определяется информационными критериями один из них – это информационный критерий Акаике, обозначается как *AIC*. Он связан с функцией правдоподобия и является информационным критерием, то есть он говорит о том насколько модель становится сложнее или проще, при добавлении или удалении из нее параметров.

$$AIC = 2k - 2\ln L,$$

где L это функция правдоподобия, то есть вероятность получить нужную выборку. В случае с линейной регрессией или методом наименьших квадратов она вычисляется намного проще поскольку имеем точное значение минимума этой функции соответственно минимум критерия Акаике. Формула выглядит так:

$$AIC = \ln \frac{RSS}{n-2} + 2k,$$

где RSS – это квадратичная ошибка, то есть сумма квадратов ошибок, которые есть у модели, а k – это число параметров, в линейной регрессии это два параметра: свободный член и коэффициент регрессии, а для полиномиальной регрессии либо ругуляризационная регрессия то количество параметров будет увеличиваться. Критерий Акаике отлично работает, когда нужно усложнить модель линейной регрессии, то есть перейти, например, к каким-то дополнительным параметрам либо к полиномиальной регрессии. То есть усложняя модель всегда можно проверить по информационному критерию насколько она стала лучше или хуже даже не ориентируюсь на её предсказательную способность. Получается, что помимо ошибки работы модели, появляется дополнительный фактор, показывающий точность работы модели и даже в том случае, когда мало данных и сложно проработать ошибку предсказание модели, остается возможность по информационному критерию выяснить точный ответ модель стала лучше или хуже.

В дополнение к информационному критерию Акаике вводят Байесовский информационный критерий, обозначается как BIC. Он схож с информационным критерием Акаике, но является менее строгим.

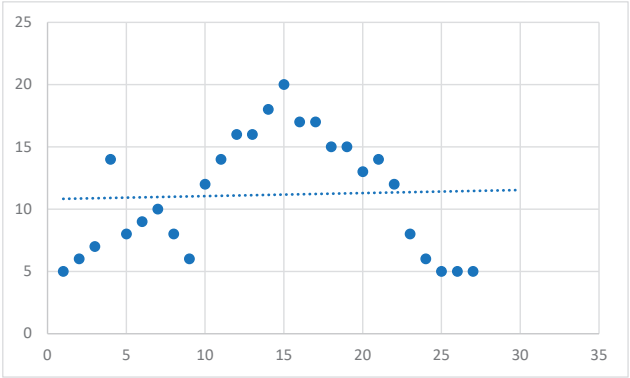
$$BIC = \ln \frac{RSS}{n-2} + k \ln n.$$

В нем также учитывается логарифм ошибки для метода наименьших квадратов, но отличается конец формулы. Если выборка достаточно большая, то ограничение получается менее строгое.

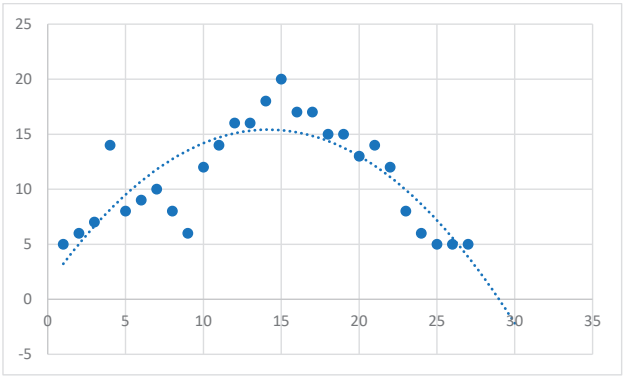
Можно сколь угодно точно предсказать наши текущие данные, однако это может потребовать большого количества параметров, и чтобы найти баланс между количеством параметров, которые нужно использовать, то есть не параметров наборов данных, которые вводятся, а гиперпараметров которые используются для того, чтобы этот набор данных преобразовать в какую-либо модель данных. Что бы не переборщить с гиперпараметрами вводится информационный критерий, минимум которого показывает оптимальное соотношение между сложность модели и её предсказательной силой.

ПОЛИНОМИАЛЬНАЯ РЕГРЕССИЯ

Одним из методов обобщения модели линейной регрессии в случае нелинейные зависимости и взаимозависимостей является применение полиномиальной регрессии. Например, есть следующий набор данных.



Конечно, можно найти прямую которая наилучшим образом опишет эти точки. Но у нее будет большая ошибка. Интуитивно понимаем, что прямая не удовлетворяет решению. Требуется механизм, описывающий кривую, которая будет обладать существенно меньшей ошибкой чем линия. Например такой:



Подход линейной регрессии может быть расширен в случае полиномиальной регрессии. То есть каждую независимую переменную считаем не в первой степени, а во второй, или даже третьей или выше. Но не всегда это имеет смысл.

Правильный подход машинного обучения включает, то, что мы знаем об окружающем мире больше, чем машина и мы можем это знание сообщить даже если машина думает по-другому. Это означает если у нас в физическом процессе, который описывается этим набором данных есть некоторые квадратичные зависимости от скорости или от температуры или может быть нелинейной, но не полиномиальная, например, корень квадратный или единица разделенная на *n* зависимость, то эту зависимость можно добавить в исходные

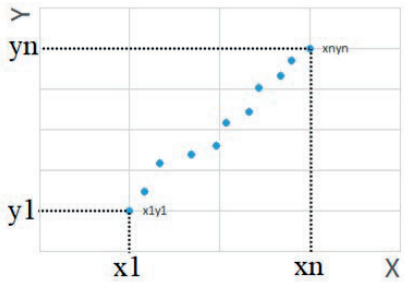
данные и как бы обогатить исходные данные этой зависимостью. И выбирать полиномиальную модель исходя из физических взаимосвязи между данными, а не фактической ошибкой, которую может минимизировать полиномиальная модель. Один из примеров модели полиномиальной регрессии будет выглядеть следующем образом:

$$y = a_0 + a_{11}x_1 + a_{22}x_2^2 + a_{34}x_3^4 + ... + a_{nk}x_n^k,$$

где k – максимальная степень полинома, значение нашего параметра которые нужно предсказывать. Такая модель полиномиальной регрессии имеет место, однако нужно принимать во внимание что она должна соотноситься с физическим миром, потому что нельзя взять произвольные степени. В большинстве случаев это будет переобучение. Используемые степени, должны иметь реальное физическое подтверждение. Хоть и при использовании высоких степеней переменных уменьшиться ошибка, уменьшится информационный критерий, и даже если модель начнёт лучше предсказывать тестовые данные это будет переобучением.

ЛИНЕАРИЗАЦИЯ РЕГРЕССИИ

Другим важным подходом к работе с нелинейными зависимостями используя линейную регрессию, является линеаризация данных, то есть замена переменных, которая позволит свести модель к линейной регрессии. Рассмотрим пример: есть некие данные сначала нужно упорядочить их по возрастанию независимого параметра x.



Отмечаем значения y_1, которое отвечает за x_1, а y_n за x_n. Далее вычисляем четыре значения – первую и последнюю точку (x_1; y_1) и (x_n; y_n). Затем по этим двум точкам считаем ряд средних, то есть среднее арифметическое, среднее геометрическое, среднее пропорциональное и в зависимости от того ближе к каким значением будут средние точки выборки можно сделать предположение о том какая связь характерна для наших данных и как можем ее использовать для линеаризации модели.

Рассмотрим какие связи могут быть и какие замены переменных могут быть использованы. Если и x, и y описываются средним арифметическим, то есть:

$$\frac{x_1 + x_n}{2} \quad \frac{y_1 + y_n}{2} \quad y = ax + b.$$

В этом случае точка будет лежать примерно, где среднее арифметическое и скорее всего модель линейной регрессии.

Если же точки хорошо описываются средним геометрическим то есть:

$$\sqrt{x_1 x_n} \quad \sqrt{y_1 y_n} \quad y = ax^b.$$

В этом случае будет явно степенной закон. Получается совершенно нелинейная модель. Однако за счёт логарифма можем её привести к виду:

$$\ln y = \ln a + b \ln x.$$

Получается, что линейность нашей модели никуда не делась, она преобразовалась в другой степенной закон.

Если возьмем, что x будет ближе к среднему арифметическому, а y ближе к среднему геометрическому, то получим уже другой степенной закон:

$$\frac{x_1 + x_n}{2} \quad \sqrt{y_1 y_n} \quad y = ab^x.$$

Это уже похоже на экспоненциальный закон, однако он также легко приводится к линейному закону.

Еще один вариант x среднее пропорциональное, а y среднее арифметическое:

$$\frac{2x_1 x_n}{x_1 + x_n} \quad \frac{y_1 + y_n}{2} \quad y = a + \frac{b}{x}.$$

При обратной ситуации:

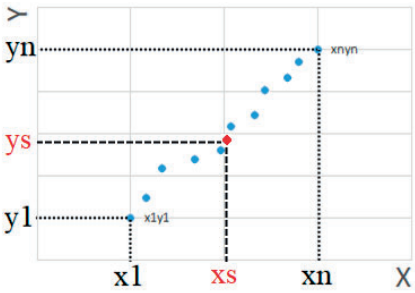
$$\frac{x_1 + x_n}{2} \quad \frac{2y_1 y_n}{y_1 + y_n} \quad y = \frac{x}{ax + b}.$$

Если x по среднему геометрическому, а y по среднему арифметическому то получится:

$$\sqrt{x_1 x_n} \quad \frac{y_1 + y_n}{2} \quad y = a + b \ln x.$$

Как принять решение какую из этих моделей выбрать. Нужно вычислить среднее значение x_s которое лежит в середине выборки, то есть число элементов делим пополам и получаем среднее значение x_s. Если оно не соответствует какому-то конкретному значению, то берём промежуточное значение между ними. Например, в выборке чётное число элементов, но нам нужно получить среднее значение x_s.

Дальше нужно от этого среднего значения найти y_s. То есть y который соответствует x_s.



После нахождения y_s, по минимуму разницы между фактическим y и вычисленным y. Находим функцию, которая лучше всего предсказывает нашу нелинейную зависимость. И потом за счёт замены переменных мы можем привести наши данные к линейной модели. Например, если у нас есть обратно пропорциональная зависимость:

$$y = a + \frac{b}{x}.$$

Можем сказать, что:

$$z + \frac{1}{x} \quad \text{и тогда} \quad y = a + bz.$$

И тогда наша модель становится линейной моделью и может быть также легко построена.

Логарифм тоже можно заменить:

$$z = \ln x \qquad y = a + bz.$$

Самое важне то, что мы можем по характеру средних значений исходных данных определить какая модель из нелинейных моделей, может лучше всего описать данные, лучше всего быть использованы в качестве замены моделей линейной регрессии. Это и есть метод линеаризации модели. Он является более простым чем метод полиномиальной регрессии. Также даёт лучший результат в плане и обобщающей способности и вычисления ошибки и вычислительной сложности.

Но он может быть применен не во всех задачах. Например, если данные распределены как-то по параболе или кубической параболе, то здесь остается полиномиальная регрессия. Однако и метод линеаризации моделей в ряде больших сложных задач машинного обучения также успешно используется.

ЧАСТЬ ПРАКТИЧЕСКИХ НАВЫКОВ К 3

ОБОГАЩЕНИЕ ДАННЫХ

Постановка задачи

Заполним отсутствующие значения по погоде интерполяционными данными.

Для точки росы вычтем температуру воздуха, направление ветра разложим на синус и косинус, для температуры воздуха вычислим первую и вторую производные. Также введем параметры по праздничным дням, дням недели, месяцам и неделям года.

Посчитаем модель линейной регрессии по первым 20 зданиям и найдем ее точность. Проверим, какой набор параметров позволяет улучшить точность.

Данные:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz
2. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
import numpy as np
from scipy.interpolate import interp1d
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

Загрузка данных, отсечение 20 зданий, объединение и оптимизация

Ввод [2]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        elif col == "timestamp":
            df[col] = pd.to_datetime(df[col])
        elif str(col_type)[:8] != "datetime":
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                            / start_mem, 1), '%)')
    return df
```

Ввод [3]:
```
buildings = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz")
weather = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz")
weather = weather[weather["site_id"] == 0]
energy = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz")
energy = energy[energy["building_id"]<20]
energy = pd.merge(left=energy, right=buildings, how="left",
                  left_on="building_id", right_on="building_id")
del buildings
print (energy.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 175680 entries, 0 to 175679
Data columns (total 9 columns):
building_id       175680 non-null int64
meter             175680 non-null int64
timestamp         175680 non-null object
meter_reading     175680 non-null float64
site_id           175680 non-null int64
primary_use       175680 non-null object
square_feet       175680 non-null int64
year_built        175680 non-null float64
floor_count       0 non-null float64
dtypes: float64(3), int64(4), object(2)
memory usage: 13.4+ MB
None
```

Интерполяция значений

Ввод [4]:
```
weather["precip_depth_1_hr"] = weather["precip_depth_1_hr"].apply(lambda x:x if x>0 else 0)
interpolate_columns = ["air_temperature", "dew_temperature",
                       "cloud_coverage", "wind_speed", "wind_direction",
                       "precip_depth_1_hr", "sea_level_pressure"]
for col in interpolate_columns:
    weather[col] = weather[col].interpolate(limit_direction='both',
                            kind='cubic')
```

Обогащение данных: погода

Ввод [5]:
```
weather["wind_direction_rad"] = weather["wind_direction"] * np.pi/180
weather["wind_direction_sin"] = np.sin(weather["wind_direction_rad"])
weather["wind_direction_cos"] = np.cos(weather["wind_direction_rad"])
weather["air_temperature_diff1"] = weather["air_temperature"].diff()
weather.at[0, "air_temperature_diff1"] = weather.at[1, "air_temperature_diff1"]
weather["air_temperature_diff2"] = weather["air_temperature_diff1"].diff()
weather.at[0, "air_temperature_diff2"] = weather.at[1, "air_temperature_diff2"]
```

Объединение погодных данных

Ввод [6]:
```
energy = energy.set_index(["timestamp", "site_id"])
weather = weather.set_index(["timestamp", "site_id"])
energy = pd.merge(left=energy, right=weather, how="left",
                  left_index=True, right_index=True)
energy.reset_index(inplace=True)
energy = energy.drop(columns=["meter", "site_id", "year_built",
                              "square_feet", "floor_count"], axis=1)
del weather
energy = reduce_mem_usage(energy)
print (energy.info())
```

```
Потребление памяти меньше на 15.41 Мб (минус 71.9 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175680 entries, 0 to 175679
Data columns (total 16 columns):
timestamp             175680 non-null datetime64[ns]
building_id           175680 non-null int8
meter_reading         175680 non-null float16
primary_use           175680 non-null category
air_temperature       175680 non-null float16
cloud_coverage        175680 non-null float16
dew_temperature       175680 non-null float16
precip_depth_1_hr     175680 non-null float16
sea_level_pressure    175680 non-null float16
```

```
wind_direction           175680 non-null float16
wind_speed               175680 non-null float16
wind_direction_rad       175680 non-null float16
wind_direction_sin       175680 non-null float16
wind_direction_cos       175680 non-null float16
air_temperature_diff1    175680 non-null float16
air_temperature_diff2    175680 non-null float16
dtypes: category(1), datetime64[ns](1), float16(13), int8(1)
memory usage: 6.0 MB
None
```

Обогащение данных: дата

Ввод [7]:
```
energy["hour"] = energy["timestamp"].dt.hour.astype("int8")
energy["weekday"] = energy["timestamp"].dt.weekday.astype("int8")
energy["week"] = energy["timestamp"].dt.week.astype("int8")
energy["month"] = energy["timestamp"].dt.month.astype("int8")
energy["date"] = pd.to_datetime(energy["timestamp"].dt.date)
dates_range = pd.date_range(start='2015-12-31', end='2017-01-01')
us_holidays = calendar().holidays(start=dates_range.min(),
                                  end=dates_range.max())
energy['is_holiday'] = energy['date'].isin(us_holidays).astype("int8")
for weekday in range(0,7):
    energy['is_wday' + str(weekday)] = energy['weekday'].isin([weekday]).astype("int8")
for week in range(1,54):
    energy['is_w' + str(week)] = energy['week'].isin([week]).astype("int8")
for month in range(1,13):
    energy['is_m' + str(month)] = energy['month'].isin([month]).astype("int8")
```

Логарифмирование данных

$$z = A * x + B * y \Rightarrow \log z = A * x + B * y \Rightarrow z =$$
$$= e \wedge Ax * e \wedge By \Rightarrow z = a \wedge x * b \wedge y.$$

Ввод [8]:
```
energy["meter_reading_log"] = np.log(energy["meter_reading"] + 1)
```

Разделение данных

Ввод [9]:
```
energy_train, energy_test = train_test_split(energy[energy["meter_reading"]>0], test_size=0.2)
print (energy_train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 86858 entries, 124905 to 130457
Data columns (total 95 columns):
timestamp                86858 non-null datetime64[ns]
building_id              86858 non-null int8
meter_reading            86858 non-null float16
primary_use              86858 non-null category
air_temperature          86858 non-null float16
cloud_coverage           86858 non-null float16
dew_temperature          86858 non-null float16
precip_depth_1_hr        86858 non-null float16
sea_level_pressure       86858 non-null float16
wind_direction           86858 non-null float16
wind_speed               86858 non-null float16
wind_direction_rad       86858 non-null float16
wind_direction_sin       86858 non-null float16
wind_direction_cos       86858 non-null float16
air_temperature_diff1    86858 non-null float16
air_temperature_diff2    86858 non-null float16
hour                     86858 non-null int8
weekday                  86858 non-null int8
week                     86858 non-null int8
```

```
month                86858 non-null int8
date                 86858 non-null datetime64[ns]
is_holiday           86858 non-null int8
is_wday0             86858 non-null int8
is_wday1             86858 non-null int8
is_wday2             86858 non-null int8
is_wday3             86858 non-null int8
is_wday4             86858 non-null int8
is_wday5             86858 non-null int8
is_wday6             86858 non-null int8
is_w1                86858 non-null int8
is_w2                86858 non-null int8
is_w3                86858 non-null int8
is_w4                86858 non-null int8
is_w5                86858 non-null int8
is_w6                86858 non-null int8
is_w7                86858 non-null int8
is_w8                86858 non-null int8
is_w9                86858 non-null int8
is_w10               86858 non-null int8
is_w11               86858 non-null int8
is_w12               86858 non-null int8
is_w13               86858 non-null int8
is_w14               86858 non-null int8
is_w15               86858 non-null int8
is_w16               86858 non-null int8
is_w17               86858 non-null int8
is_w18               86858 non-null int8
is_w19               86858 non-null int8
is_w20               86858 non-null int8
is_w21               86858 non-null int8
is_w22               86858 non-null int8
is_w23               86858 non-null int8
is_w24               86858 non-null int8
is_w25               86858 non-null int8
is_w26               86858 non-null int8
is_w27               86858 non-null int8
is_w28               86858 non-null int8
is_w29               86858 non-null int8
is_w30               86858 non-null int8
is_w31               86858 non-null int8
is_w32               86858 non-null int8
is_w33               86858 non-null int8
is_w34               86858 non-null int8
is_w35               86858 non-null int8
is_w36               86858 non-null int8
is_w37               86858 non-null int8
is_w38               86858 non-null int8
is_w39               86858 non-null int8
is_w40               86858 non-null int8
is_w41               86858 non-null int8
is_w42               86858 non-null int8
is_w43               86858 non-null int8
is_w44               86858 non-null int8
is_w45               86858 non-null int8
is_w46               86858 non-null int8
is_w47               86858 non-null int8
is_w48               86858 non-null int8
is_w49               86858 non-null int8
is_w50               86858 non-null int8
is_w51               86858 non-null int8
is_w52               86858 non-null int8
is_w53               86858 non-null int8
is_m1                86858 non-null int8
is_m2                86858 non-null int8
is_m3                86858 non-null int8
is_m4                86858 non-null int8
is_m5                86858 non-null int8
is_m6                86858 non-null int8
```

```
is_m7                   86858 non-null int8
is_m8                   86858 non-null int8
is_m9                   86858 non-null int8
is_m10                  86858 non-null int8
is_m11                  86858 non-null int8
is_m12                  86858 non-null int8
meter_reading_log       86858 non-null float16
dtypes: category(1), datetime64[ns](2), float16(14), int8(78)
memory usage: 10.9 MB
None
```

Линейная регрессия

Ввод [12]:
```
hours = range(0, 24)
buildings = range(0, energy_train["building_id"].max() + 1)
lr_columns = ["meter_reading_log", "hour", "building_id",
              "air_temperature", "dew_temperature",
              "sea_level_pressure", "wind_speed", "cloud_coverage",
              "air_temperature_diff1", "air_temperature_diff2",
              "is_holiday"]
for wday in range(0,7):
    lr_columns.append("is_wday" + str(wday))
for week in range(1,54):
    lr_columns.append("is_w" + str(week))
for month in range(1,13):
    lr_columns.append("is_m" + str(month))
energy_train_lr = pd.DataFrame(energy_train, columns=lr_columns)
energy_lr = [[]]*len(buildings)
for building in buildings:
    energy_lr[building] = [[]]*len(hours)
    energy_train_b = energy_train_lr[energy_train_lr["building_id"]==building]
    for hour in hours:
        energy_train_bh = energy_train_b[energy_train_b["hour"]==hour]
        y = energy_train_bh["meter_reading_log"]
        x = energy_train_bh.drop(labels=["meter_reading_log",
            "hour", "building_id"], axis=1)
        model = LinearRegression(fit_intercept=False).fit(x, y)
        energy_lr[building][hour] = model.coef_
        energy_lr[building][hour] = np.append(energy_lr[building][hour], model.intercept_)
print (energy_lr[0])
```

```
[array([ 4.73197084e-03,  8.93366151e-03, -3.08676995e-03,  2.94966809e-03,
        -9.89266112e-03,  1.67482495e-02, -1.62267014e-02, -8.47926289e-02,
        -3.66255641e-04,  4.00551033e+00,  3.99231124e+00,  4.03821278e+00,
         3.99159217e+00,  3.98330736e+00,  3.99078774e+00,  3.98518777e+00,
        -2.21133232e-05, -3.94284725e-05,  1.90734863e-05,  5.45382500e-06,
         1.94907188e-05, -4.59551811e-05,  1.97291374e-05,  2.23517418e-05,
        -1.68085098e-05,  4.64916229e-06,  1.14440918e-05,  5.36441803e-06,
         1.62124634e-05, -4.76837158e-07,  1.08480453e-05, -5.00679016e-06,
        -5.00679016e-06,  2.38418579e-07,  6.91413879e-06,  1.13830042e+00,
         8.77157927e-01,  8.31271648e-01,  9.50120926e-01,  1.03997469e+00,
         8.66564512e-01,  8.54584038e-01,  7.46237993e-01,  1.02058601e+00,
         1.00669742e+00,  9.65929985e-01,  9.88794327e-01,  1.01486206e+00,
         1.06163073e+00,  8.63721371e-01,  8.47085357e-01,  8.47538471e-01,
         8.05390775e-01,  8.16756308e-01,  8.55287671e-01,  8.34055901e-01,
         8.77578676e-01,  8.64419580e-01,  7.73206174e-01,  7.43238807e-01,
         7.28382885e-01,  7.01682091e-01,  9.14781392e-01,  6.59618020e-01,
         2.43507624e-01,  6.13937378e-01,  7.17874765e-01,  9.17946935e-01,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  3.28813291e+00,  3.48118162e+00,  3.52674222e+00,
         3.52249241e+00,  3.54438639e+00,  3.55073547e+00,  3.54706335e+00,
         3.52867436e+00,  0.00000000e+00]), array([ 6.79896725e-03,  6.53352030e-03, -6.02531806e-03,  4.73713176e-03,
        -1.41378120e-02,  8.87155533e-04,  2.59605795e-02, -2.91809291e-02,
        -7.59439543e-04,  5.41822195e+00,  5.42371321e+00,  5.47761822e+00,
         5.42535496e+00,  5.40479708e+00,  5.41514444e+00,  5.40463114e+00,
         7.78436661e-05, -1.35123730e-04, -1.67489052e-05, -1.46627426e-05,
        -4.43458557e-05,  1.15275383e-04,  3.73125076e-05, -7.87377357e-05,
         1.87158585e-05, -3.14712524e-05,  1.35898590e-05, -2.49147415e-05,
         8.10623169e-06,  1.59740448e-05,  1.59740448e-05, -1.43051147e-06,
        -5.96046448e-06,  2.86102295e-06,  3.81469727e-06,  1.53577507e+00,
         1.25905335e+00,  1.17573225e+00,  1.28773880e+00,  1.33054483e+00,
         1.17892170e+00,  1.19457555e+00,  1.24039626e+00,  1.34659266e+00,
         1.28691447e+00,  1.20806468e+00,  1.43025279e+00,  1.38179278e+00,
         1.49075770e+00,  1.22951555e+00,  1.20409834e+00,  1.18198085e+00,
         1.16828239e+00,  1.09574103e+00,  1.21207213e+00,  1.12137485e+00,
         1.21801543e+00,  1.04258060e+00,  9.82888222e-01,  1.00062323e+00,
         9.40358162e-01,  8.18687320e-01,  1.04725337e+00,  8.90549898e-01,
         6.34157419e-01,  6.54630005e-01,  1.04451382e+00,  1.13697433e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
```

```
 0.00000000e+00,  4.48479176e+00,  4.71998501e+00,  4.83637905e+00,
 4.67153311e+00,  4.76615953e+00,  4.81553078e+00,  4.84177351e+00,
 4.82512712e+00,  0.00000000e+00]), array([ 1.11181941e-02,  4.90948791e-03, -5.01786917e-03,  2.38450244e-04,
-1.02477446e-02,  1.65177733e-02, -4.71105054e-03, -5.11235148e-02,
 1.51932612e-03,  4.91280079e+00,  4.90867233e+00,  4.94168472e+00,
 4.91513586e+00,  4.89400673e+00,  4.91732740e+00,  4.88930655e+00,
 1.66381360e-04, -6.79194927e-05,  4.10974026e-05, -6.57439232e-05,
 1.13517046e-04,  3.77595425e-05, -8.56518745e-05, -4.44650650e-05,
-4.64916229e-05,  7.99894333e-05,  1.88350677e-05, -1.71065331e-05,
 5.03063202e-05,  4.38690186e-05,  1.46031380e-05, -2.20537186e-06,
-1.23977661e-05,  1.70469284e-05,  1.09672546e-05,  1.33476150e+00,
 1.14938354e+00,  9.97875929e-01,  1.13531375e+00,  1.23628759e+00,
 1.10208261e+00,  1.08507431e+00,  9.97412920e-01,  1.23069537e+00,
 1.23123753e+00,  1.17090189e+00,  1.27068377e+00,  1.26712394e+00,
 1.33408332e+00,  1.14405632e+00,  1.07418442e+00,  1.10967827e+00,
 1.06303525e+00,  1.04514432e+00,  1.14950275e+00,  1.05138254e+00,
 1.11161172e+00,  1.02901483e+00,  1.02221179e+00,  8.60333800e-01,
 7.97271013e-01,  7.47336030e-01,  9.49933589e-01,  7.78840601e-01,
 5.01694977e-01,  5.30237496e-01,  8.98800254e-01,  9.68578875e-01,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  4.05602884e+00,  4.27213955e+00,  4.34010220e+00,
 4.25213528e+00,  4.28530502e+00,  4.31097031e+00,  4.42427540e+00,
 4.43776417e+00,  0.00000000e+00]), array([ 1.44999586e-02,  5.82565181e-03,  6.28530979e-05,  4.51809913e-03,
-1.62372440e-02,  2.87999026e-02, -1.72969326e-02, -1.01071008e-01,
 1.90086383e-03,  2.41714239e+00,  2.38201261e+00,  2.40509009e+00,
 2.39122558e+00,  2.39063430e+00,  2.40253592e+00,  2.38640022e+00,
-1.21142715e-04,  6.25848770e-07,  6.75693154e-05, -1.34557486e-05,
-2.48327851e-05, -2.75373459e-05,  3.33189964e-05, -2.73585320e-05,
-4.79817390e-05, -6.19888306e-06, -7.15255737e-06,  1.69873238e-06,
-1.68681145e-05, -1.36345625e-05, -2.45571136e-05,  8.58306885e-06,
-2.38418579e-06,  7.80820847e-06,  7.15255737e-07,  7.18853772e-01,
 5.10547936e-01,  4.47982609e-01,  5.61804175e-01,  6.94607794e-01,
 4.83092546e-01,  5.38829029e-01,  4.26187158e-01,  6.36447430e-01,
 5.93891740e-01,  6.09942794e-01,  6.08018219e-01,  5.30451417e-01,
 6.08516037e-01,  4.48966265e-01,  6.08749449e-01,  6.58343196e-01,
 5.80681622e-01,  6.02801085e-01,  6.38177991e-01,  5.89373410e-01,
 6.54496908e-01,  5.32085657e-01,  4.94068921e-01,  4.48220044e-01,
 4.17396516e-01,  3.76355946e-01,  4.51746106e-01,  3.42692465e-01,
-9.38556194e-02,  8.60459805e-02,  4.58292216e-01,  5.00897765e-01,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  1.96592581e+00,  2.12048078e+00,  2.13651037e+00,
 2.19807816e+00,  2.02090526e+00,  2.06059313e+00,  2.14308739e+00,
 2.12766743e+00,  0.00000000e+00]), array([ 3.92469727e-02, -1.29195126e-02, -8.20099749e-03, -2.15508454e-02,
-8.21736176e-03,  7.72886723e-02, -4.18086722e-02, -1.33545697e-01,
 2.30970979e-03,  6.36716032e+00,  6.39545584e+00,  6.43821716e+00,
 6.36810398e+00,  6.41121197e+00,  6.29648972e+00,  6.18680859e+00,
-1.18538737e-04, -1.33514404e-04, -7.77393579e-05,  8.28877091e-05,
-6.66230917e-05, -1.08927488e-05,  5.18113375e-05,  8.98279250e-05,
-3.29986215e-05,  9.48980451e-05,  3.27527523e-05, -1.17070973e-04,
 8.93175602e-05, -8.20755959e-05, -2.80737877e-05,  5.84125519e-05,
-1.64866447e-04,  5.99622726e-05,  7.15255737e-07,  1.94785690e+00,
 1.67788982e+00,  1.44920468e+00,  1.44999647e+00,  1.55744982e+00,
 1.42721486e+00,  1.35481215e+00,  1.14488220e+00,  1.40235853e+00,
 1.40866899e+00,  1.32631826e+00,  1.37644923e+00,  1.26397848e+00,
 1.34922111e+00,  1.16080987e+00,  1.26737571e+00,  1.35233116e+00,
 1.24327517e+00,  1.24583948e+00,  1.32703328e+00,  1.22691679e+00,
 1.40218270e+00,  1.28222501e+00,  1.34575438e+00,  1.35876954e+00,
 1.34021139e+00,  1.27057838e+00,  1.43966603e+00,  1.25613356e+00,
 9.63081539e-01,  8.45470250e-01,  1.45440960e+00,  1.54499054e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  5.04892015e+00,  5.50810242e+00,  5.58990622e+00,
 5.72637367e+00,  5.62464142e+00,  5.74118805e+00,  5.57836723e+00,
 5.64588594e+00,  0.00000000e+00]), array([ 2.91250609e-02, -8.40237271e-03, -9.25683603e-03, -5.18124923e-03,
-2.27246247e-03, -5.03622107e-02,  1.14788115e-02,  1.34514943e-02,
-4.67464328e-04,  7.07167339e+00,  7.13805485e+00,  7.15160894e+00,
 7.08468866e+00,  7.10024691e+00,  7.09320927e+00,  7.08298874e+00,
 1.18345022e-04,  7.17937946e-05,  1.32381916e-04,  2.82390046e+00,
-2.45526433e-04, -1.21384859e-04,  2.00867653e-05,  2.34842300e-05,
 2.29835510e-04,  1.60545111e-04, -5.01275063e-05,  1.01424754e-04,
-2.89529562e-05, -3.75509262e-06,  1.30861998e-04, -5.67734241e-05,
-1.77621841e-05, -9.15527344e-05, -2.53915787e-05,  1.81497812e+00,
 1.45523715e+00,  1.59960198e+00,  1.47157884e+00,  1.65337038e+00,
 1.52617693e+00,  1.63008809e+00,  1.26427329e+00,  1.62174153e+00,
 1.62539458e+00,  1.57613635e+00,  1.76133561e+00,  1.72482252e+00,
 1.93913507e+00,  1.61353672e+00,  1.55759263e+00,  1.70084047e+00,
 2.02033949e+00,  1.79077923e+00,  1.61142898e+00,  1.23755360e+00,
 1.27122819e+00,  1.40526140e+00,  1.29272211e+00,  1.20533347e+00,
 8.85823667e-01,  1.17306828e+00,  1.05962598e+00,  9.53606904e-01,
 4.70136166e-01,  7.98422098e-01,  1.07319880e+00,  1.09983647e+00,
 0.00000000e+00,  2.82345557e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  5.63638449e+00,  5.80642939e+00,  5.79611969e+00,
 5.65599823e+00,  5.55575228e+00,  6.00536013e+00,  6.13066578e+00,
 6.29904318e+00,  0.00000000e+00]), array([ 3.62274908e-02,  4.21422441e-03, -4.55738977e-03,  1.74289569e-03,
-1.89797599e-02, -4.83200699e-02,  2.11136267e-02, -1.76892087e-01,
-1.05516613e-03,  4.45730543e+00,  4.40103388e+00,  4.45277500e+00,
 4.43969297e+00,  4.41280127e+00,  4.41401005e+00,  4.42931509e+00,
-1.13613904e-04,  6.20540231e-06,  6.73979521e-05, -9.63807106e-05,
-1.63376331e-04, -9.93907452e-05,  6.56843185e-05, -6.92009926e-05,
 8.29696655e-05,  4.76837158e-05, -2.02059746e-04, -4.74452972e-05,
-5.65052032e-05,  1.08599663e-04, -1.47819519e-04, -4.48226929e-05,
```

```
 4.19616699e-05, -5.72204590e-06,  1.23977661e-05,  1.16609645e+00,
 1.02827275e+00,  9.69598949e-01,  1.08870208e+00,  1.19509697e+00,
 9.89225984e-01,  1.04445124e+00,  7.28001595e-01,  1.05503690e+00,
 1.02603853e+00,  1.00651336e+00,  9.50502157e-01,  9.47727501e-01,
 9.91525173e-01,  8.43613386e-01,  8.44007492e-01,  8.61444533e-01,
 7.84709930e-01,  8.30325007e-01,  8.69281054e-01,  8.88422132e-01,
 1.07753789e+00,  9.30879354e-01,  1.01015782e+00,  8.89324367e-01,
 1.02956045e+00,  8.98827791e-01,  1.14837813e+00,  8.61912489e-01,
 5.12979507e-01,  4.71070766e-01,  1.03039885e+00,  1.03757739e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  3.67659688e+00,  3.77612090e+00,  3.84462452e+00,
 3.93382072e+00,  3.98931170e+00,  3.90677977e+00,  3.91636300e+00,
 3.96370077e+00,  0.00000000e+00]), array([ 6.57235563e-04,  1.27943046e-02, -3.96213308e-03,  9.69708711e-03,
-9.45654511e-03,  4.29192446e-02, -1.53402165e-02,  9.73137841e-02,
-1.22608244e-03,  4.41996145e+00,  4.41980886e+00,  4.48604584e+00,
 4.41285801e+00,  4.42377138e+00,  4.43143940e+00,  4.37715435e+00,
-2.20946968e-04, -2.33039260e-04, -1.53573230e-04,  1.78031623e-04,
 9.82619822e-05, -9.29161906e-05,  3.78340483e-05,  1.43095851e-04,
-7.01099634e-05, -4.29630280e-04,  7.20471144e-06, -2.40854919e-04,
 1.57549977e-04,  5.28842211e-05, -2.39461660e-05, -2.37226486e-05,
-1.07288361e-05,  3.21865082e-05, -1.45435333e-05,  1.22239137e+00,
 1.25961566e+00,  9.83165026e-01,  9.36649680e-01,  1.02880740e+00,
 9.83462095e-01,  9.17870641e-01,  7.90732622e-01,  1.11373782e+00,
 1.08155191e+00,  9.93509531e-01,  1.24940002e+00,  1.30465233e+00,
 1.30945337e+00,  1.19803846e+00,  1.11624837e+00,  9.42322791e-01,
 8.85850489e-01,  9.94923353e-01,  9.61307824e-01,  8.55672002e-01,
 9.47718740e-01,  8.73285115e-01,  7.66999006e-01,  7.99081326e-01,
 8.03737819e-01,  6.52087569e-01,  8.30030322e-01,  5.79815507e-01,
 3.35221052e-01,  4.98025835e-01,  8.00160468e-01,  9.40027952e-01,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  3.52190828e+00,  3.86395788e+00,  3.93030882e+00,
 3.74782944e+00,  3.87486124e+00,  3.96408367e+00,  4.04051065e+00,
 4.01844883e+00,  0.00000000e+00]), array([-4.69355704e-03,  1.22937802e-02,  1.50875375e-03, -1.97738409e-04,
-6.31751120e-03,  9.26132500e-03,  3.45832855e-03, -2.84738541e-02,
 1.34873390e-03,  1.82626379e+00,  1.83001554e+00,  1.85963702e+00,
 1.84346437e+00,  1.85405958e+00,  1.87685895e+00,  1.87023854e+00,
 1.69873238e-06,  3.71485949e-05, -3.93241644e-05,  2.14999110e-01,
 4.72664833e-05,  1.67042017e-05,  5.62220812e-05,  1.75535679e-05,
 6.71446323e-05, -8.10604542e-05, -1.15530565e-05,  6.76140189e-06,
 3.00956890e-06, -4.29153442e-06,  1.11907721e-05,  4.94718552e-06,
-1.72853470e-05,  7.56978989e-06, -7.62939453e-06,  6.42774820e-01,
 3.33621085e-01,  1.98692322e-01,  3.13531786e-01,  4.59396303e-01,
 3.48566175e-01,  3.15869570e-01,  3.61969501e-01,  6.08340144e-01,
 6.10047519e-01,  5.97437680e-01,  5.70413351e-01,  6.04956508e-01,
 5.62975764e-01,  4.21342075e-01,  3.70388508e-01,  4.17405069e-01,
 3.58556271e-01,  3.70896339e-01,  4.21873391e-01,  4.55970556e-01,
 4.76107150e-01,  5.40339291e-01,  2.87635922e-01,  2.97080845e-01,
 2.23896265e-01,  2.66753793e-01,  3.04135323e-01,  2.68402368e-01,
 3.23134065e-02,  1.03799999e-02,  2.67327935e-01,  4.17391539e-01,
 0.00000000e+00,  2.14985132e-01,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  1.51350749e+00,  1.67517805e+00,  1.60298514e+00,
 1.62163019e+00,  1.58597279e+00,  1.57833803e+00,  1.60575533e+00,
 1.56016660e+00,  0.00000000e+00]), array([ 1.52347062e-03, -7.60033727e-05, -1.08784642e-02, -1.39014684e-02,
-5.91915846e-03,  8.89068544e-02, -1.99669898e-02, -6.18733689e-02,
-8.17322731e-03,  7.92009306e+00,  7.97467232e+00,  7.93432713e+00,
 7.89827061e+00,  7.94518709e+00,  7.90196466e+00,  7.88853693e+00,
 9.48086381e-05,  3.50475311e-05,  2.18302011e-06, -1.19134784e-05,
-1.21474266e-04,  1.36345625e-04,  2.55018473e-04,  1.58540905e-04,
 2.56374478e-05, -9.32067633e-06,  7.83354044e-05, -1.75282359e-04,
-1.61230564e-04,  8.69929790e-05, -1.80542469e-04, -1.77979469e-04,
-2.46167183e-05,  5.75184822e-05,  1.07765198e-04,  2.27700543e+00,
 1.71055698e+00,  1.74598312e+00,  1.72416067e+00,  1.83322167e+00,
 1.88187838e+00,  1.64326906e+00,  1.81576180e+00,  1.91125321e+00,
 1.95699453e+00,  1.90138328e+00,  1.92495787e+00,  1.96615291e+00,
 1.88291645e+00,  1.70159030e+00,  1.68420744e+00,  1.63282657e+00,
 1.59050632e+00,  1.53779495e+00,  1.59056461e+00,  1.55412269e+00,
 1.69269001e+00,  1.72966242e+00,  1.47611690e+00,  1.41349149e+00,
 1.47323823e+00,  1.53400326e+00,  1.55109859e+00,  1.47111893e+00,
 1.39411902e+00,  1.25964141e+00,  1.49586940e+00,  1.50367475e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  6.59382200e+00,  6.88741493e+00,  6.92138720e+00,
 6.96353006e+00,  6.99688482e+00,  7.06219721e+00,  7.00287342e+00,
 7.03550529e+00,  0.00000000e+00]), array([ 1.98022798e-02, -1.43291000e-02, -1.58117153e-02, -7.10809231e-03,
-4.02095914e-03,  8.28556716e-02, -2.96297297e-02, -3.17357257e-02,
 4.94652018e-02,  1.01922455e+01,  1.02496014e+01,  1.03582563e+01,
 1.02315426e+01,  1.02705154e+01,  1.02398148e+01,  1.02297421e+01,
-1.88618898e-04,  5.21503389e-05,  3.70375812e-04, -1.62746757e-04,
-4.02866630e-04, -1.14321709e-04,  2.67267227e-04, -1.02162361e-04,
 2.01880932e-04,  6.16014004e-05,  1.04844570e-04,  2.46308744e-04,
 2.16573477e-04, -1.08540058e-04, -1.20043755e-04,  4.35709953e-05,
 1.55448914e-04, -8.70227814e-06, -8.98838043e-05,  2.69642925e+00,
 2.32795906e+00,  2.23817110e+00,  2.26914287e+00,  2.46441031e+00,
 2.45153451e+00,  2.29869223e+00,  2.26905012e+00,  2.49498916e+00,
 2.45307541e+00,  2.44253874e+00,  2.17222595e+00,  2.17476892e+00,
 2.22145772e+00,  2.03092241e+00,  1.98852789e+00,  2.08043361e+00,
 2.01186037e+00,  2.01675773e+00,  2.04431200e+00,  2.00701189e+00,
 2.23802614e+00,  2.15566635e+00,  1.89717197e+00,  1.92661071e+00,
 1.90226126e+00,  2.05671191e+00,  2.09524822e+00,  2.12998915e+00,
 1.71381760e+00,  1.90691686e+00,  2.32590532e+00,  2.27451563e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
```

```
 0.00000000e+00,  8.68207741e+00,  8.80596352e+00,  8.91505814e+00,
 9.26327419e+00,  9.16799927e+00,  9.07487297e+00,  9.03799057e+00,
 8.82403660e+00,  0.00000000e+00]), array([ 3.41834947e-02, -3.12851444e-02, -1.53376367e-02, -1.07624531e-02,
 1.54690593e-02,  6.64699972e-02, -5.47680855e-02, -3.41198593e-03,
-3.94055992e-03,  1.00296593e+01,  1.00410738e+01,  9.97793388e+00,
 1.00211525e+01,  1.00335340e+01,  9.95054722e+00,  9.98056602e+00,
 3.36237252e-04,  3.18720937e-04, -1.32903457e-04, -8.93175602e-05,
-1.30757689e-04, -2.08660960e-04, -1.01476908e-04,  5.44190407e-05,
-2.28263438e-04,  2.71007419e-04,  5.26905060e-05, -3.20717692e-04,
-3.30388546e-04, -2.12427229e-04, -8.62479210e-05,  1.48117542e-05,
-1.71542168e-04, -7.28368759e-05,  1.02162361e-04,  2.49870157e+00,
 2.22006941e+00,  2.03737640e+00,  2.23756599e+00,  2.47871447e+00,
 2.44166732e+00,  2.26587629e+00,  2.30582833e+00,  2.45617390e+00,
 2.47314692e+00,  2.46842337e+00,  2.39956617e+00,  2.39881420e+00,
 2.49755335e+00,  2.31164122e+00,  2.10698271e+00,  2.18229079e+00,
 2.06473684e+00,  2.13393044e+00,  2.04982996e+00,  1.99510407e+00,
 2.16673064e+00,  2.09762669e+00,  1.85496712e+00,  1.86827099e+00,
 1.70318699e+00,  1.81178474e+00,  1.90606296e+00,  1.82257056e+00,
 1.29367137e+00,  1.45845711e+00,  2.02673769e+00,  2.00105119e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  8.57678509e+00,  8.60708046e+00,  8.69919109e+00,
 8.71796322e+00,  8.81614590e+00,  8.85441971e+00,  8.90074348e+00,
 8.86229324e+00,  0.00000000e+00]), array([ 3.20700407e-02, -1.93958059e-02, -2.07150001e-02, -4.32491712e-02,
-3.10366675e-02, -1.16733536e-01,  7.90066868e-02, -1.17292643e-01,
-1.23967230e-03,  1.27129669e+01,  1.25826797e+01,  1.27234926e+01,
 1.26822052e+01,  1.26950245e+01,  1.25536594e+01,  1.27560720e+01,
 6.65485859e-05, -2.59980559e-04, -1.99452043e-04,  3.84986401e-04,
 1.25382096e-04,  1.35511160e-04,  6.52968884e-05, -7.50124454e-05,
-1.38401985e-04, -1.30477361e-04, -4.67821956e-05, -2.13503838e-04,
 9.15527344e-05,  1.33991241e-04,  3.02791595e-04, -2.47955322e-05,
-2.25067139e-04, -1.52587891e-05,  2.23159790e-04,  2.68653965e+00,
 2.66635370e+00,  2.41493917e+00,  3.15009975e+00,  3.18736362e+00,
 3.20964527e+00,  3.03268600e+00,  2.92462921e+00,  3.17229581e+00,
 3.16878414e+00,  3.03762031e+00,  2.72871685e+00,  2.78229094e+00,
 2.71897316e+00,  2.55614352e+00,  2.41884351e+00,  2.44657612e+00,
 2.41101599e+00,  2.34568501e+00,  2.32411242e+00,  2.69347906e+00,
 2.89575052e+00,  2.82449222e+00,  2.71667719e+00,  2.50790381e+00,
 2.64978361e+00,  2.56925440e+00,  2.76390934e+00,  2.39247656e+00,
 2.06698132e+00,  2.32963991e+00,  2.48644161e+00,  2.42403221e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  1.11526051e+01,  1.08499613e+01,  1.06473932e+01,
 1.13645620e+01,  1.13684921e+01,  1.11305742e+01,  1.09339142e+01,
 1.12576923e+01,  0.00000000e+00]), array([ 5.08515052e-02, -2.96361707e-02, -7.00910017e-03, -2.15096399e-02,
-1.19051104e-02, -8.48632306e-02,  4.46785204e-02,  7.62596428e-02,
 2.74333730e-03,  5.80736589e+00,  5.86611080e+00,  5.86828327e+00,
 5.80796957e+00,  5.84264517e+00,  5.79274321e+00,  5.78648472e+00,
-1.24245882e-04,  7.14004040e-04, -1.28835440e-04, -1.53712928e-04,
-2.29984522e-04, -4.11272049e-06, -2.88188457e-05, -1.34408474e-04,
-3.46302986e-05,  1.73747540e-05,  3.61204147e-05, -3.70740891e-05,
-5.48362732e-06,  9.53674316e-06,  2.86102295e-06, -1.43051147e-06,
-1.43051147e-06,  1.43051147e-06,  4.76837158e-07,  1.88613796e+00,
 1.49400115e+00,  1.18741858e+00,  1.23866451e+00,  1.34508872e+00,
 1.30761313e+00,  1.23522580e+00,  1.22009659e+00,  1.35891283e+00,
 1.40448165e+00,  1.37054455e+00,  1.36987782e+00,  1.40854239e+00,
 1.36139560e+00,  1.19358468e+00,  1.12292182e+00,  1.13455856e+00,
 1.15352654e+00,  1.10652614e+00,  1.18417656e+00,  1.13387585e+00,
 1.23252916e+00,  1.21823120e+00,  1.00060058e+00,  1.25755048e+00,
 1.18627942e+00,  1.27117705e+00,  1.26715112e+00,  1.15140998e+00,
 6.88769996e-01,  9.47196007e-01,  1.20807803e+00,  1.14442265e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  4.77740145e+00,  5.12800646e+00,  5.14434481e+00,
 5.19769955e+00,  5.20675707e+00,  5.20579433e+00,  4.99067593e+00,
 5.10126209e+00,  0.00000000e+00]), array([ 2.69463863e-02, -3.23130284e-03, -1.49888042e-02,  6.51037693e-03,
-2.83567123e-02, -5.54083996e-02,  1.47164613e-03, -3.73499766e-02,
-9.68574546e-03,  9.70586586e+00,  9.77675247e+00,  9.71123505e+00,
 9.69422531e+00,  9.70898533e+00,  9.67444611e+00,  9.67058372e+00,
-2.98768282e-06,  6.02826476e-05, -1.68249011e-04,  8.83862376e-05,
 1.10227615e-04, -9.31322575e-06,  1.60228461e-04,  4.03113663e-05,
 9.36239958e-05,  1.46597624e-04, -1.46988779e-04,  2.17318535e-04,
-3.79681587e-04, -2.44379044e-05,  2.50339508e-05,  4.00543213e-05,
-1.62363052e-04,  2.55584717e-04,  4.38690186e-05,  2.73150706e+00,
 2.14215326e+00,  2.09984946e+00,  2.24180555e+00,  2.18108201e+00,
 2.22739744e+00,  2.19767904e+00,  2.13957644e+00,  2.35147953e+00,
 2.33782673e+00,  2.30797672e+00,  2.24685240e+00,  2.25987434e+00,
 2.26856089e+00,  2.05765390e+00,  1.91013265e+00,  1.84728587e+00,
 1.90011728e+00,  1.84607482e+00,  1.94378340e+00,  1.75262451e+00,
 2.01328993e+00,  1.92169929e+00,  1.88190508e+00,  2.07936502e+00,
 2.05106091e+00,  2.12197328e+00,  2.09840560e+00,  1.88127303e+00,
 1.31969833e+00,  1.72499716e+00,  1.96545744e+00,  1.89111781e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  8.18045235e+00,  8.38146305e+00,  8.39600182e+00,
 8.50786972e+00,  8.66379642e+00,  8.67814445e+00,  8.44900513e+00,
 8.68440056e+00,  0.00000000e+00]), array([ 3.18146236e-02, -1.15176849e-02, -1.01363435e-02, -2.48695537e-03,
-2.55778581e-02, -7.09947050e-02,  4.28656377e-02,  2.87366658e-03,
-2.54466711e-03,  7.35278988e+00,  7.38449478e+00,  7.40757418e+00,
 7.34611940e+00,  7.34180975e+00,  7.32249928e+00,  7.29408932e+00,
-1.64955854e-04,  3.08156013e-04, -1.14172697e-04, -2.37464905e-04,
 2.92062759e-06,  1.63346529e-04, -1.20848417e-04,  1.11460686e-04,
 6.29425049e-05,  1.10387802e-04, -5.79357147e-05, -9.72747803e-05,
```

```
-1.04367733e-04,  8.35657120e-05, -1.23977661e-05,  2.19345093e-05,
 2.86102295e-06,  1.57356262e-05,  0.00000000e+00,  2.17701602e+00,
 1.78694165e+00,  1.62056172e+00,  1.66065168e+00,  1.57702351e+00,
 1.71351981e+00,  1.63332391e+00,  1.51748037e+00,  1.70760441e+00,
 1.73750699e+00,  1.72216761e+00,  1.88365030e+00,  1.94986033e+00,
 1.88775730e+00,  1.67033231e+00,  1.62684751e+00,  1.55336452e+00,
 1.56591749e+00,  1.56815338e+00,  1.54705763e+00,  1.45272207e+00,
 1.61369562e+00,  1.52340126e+00,  1.33015299e+00,  1.34103608e+00,
 1.29024982e+00,  1.37207079e+00,  1.39544868e+00,  1.27873981e+00,
 8.18364084e-01,  1.17747378e+00,  1.38632607e+00,  1.36779690e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  6.02372217e+00,  6.38618326e+00,  6.44875765e+00,
 6.34875536e+00,  6.45701694e+00,  6.54952097e+00,  6.59560394e+00,
 6.64535046e+00,  0.00000000e+00]), array([ 3.07816360e-02, -1.46558080e-02, -9.37638711e-03, -6.66692853e-03,
-1.70300528e-02, -4.10860255e-02,  1.12255439e-02, -1.31944474e-02,
 3.72741371e-04,  6.94802904e+00,  6.99268723e+00,  7.03362417e+00,
 6.94886971e+00,  6.95143700e+00,  6.95718622e+00,  6.94198322e+00,
-1.44213438e-04,  5.80474734e-05,  9.93460417e-05,  5.28506935e-05,
-9.56654549e-06,  2.31415033e-05, -6.82175159e-05, -1.99377537e-04,
 1.29938126e-05, -2.48551369e-05,  1.16407871e-04, -2.84612179e-06,
-6.52670860e-05,  1.16109848e-04, -3.95476818e-05,  1.87069178e-04,
 5.37633896e-05,  3.24010849e-04, -7.20024109e-05,  2.14617491e+00,
 1.65537739e+00,  1.49189496e+00,  1.57865620e+00,  1.49664140e+00,
 1.63257504e+00,  1.51634884e+00,  1.45941663e+00,  1.66081035e+00,
 1.65835822e+00,  1.66272783e+00,  1.80986428e+00,  1.86442745e+00,
 1.80699813e+00,  1.64252543e+00,  1.57325220e+00,  1.47216821e+00,
 1.47295916e+00,  1.42419708e+00,  1.44344413e+00,  1.36298513e+00,
 1.56687570e+00,  1.45581913e+00,  1.21952891e+00,  1.17172587e+00,
 1.14573157e+00,  1.22700262e+00,  1.24074650e+00,  1.20045435e+00,
 8.36321831e-01,  1.16349959e+00,  1.37778962e+00,  1.33739161e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  5.74486446e+00,  6.08481216e+00,  6.12825012e+00,
 6.01925755e+00,  6.14873791e+00,  6.18910789e+00,  6.25221777e+00,
 6.20590401e+00,  0.00000000e+00]), array([ 3.08428146e-02, -1.44077353e-02, -8.35567806e-03, -3.28034908e-03,
 1.44094229e-03, -6.62288964e-02,  3.83934900e-02, -9.10568386e-02,
-1.78240240e-04,  6.62980127e+00,  6.63267756e+00,  6.69877481e+00,
 6.63848543e+00,  6.59410524e+00,  6.58385372e+00,  6.58492851e+00,
-2.68518925e-05, -4.10139561e-04, -1.89483166e-04,  2.79627872e+00,
 4.76837158e-07,  1.43289566e-04,  7.53402710e-05,  1.65760517e-04,
-1.33931637e-04,  7.26580620e-05,  1.01327896e-05,  4.23789024e-05,
 7.87973404e-05, -9.26256180e-05, -6.67572021e-06,  4.76837158e-06,
-1.43051147e-06, -3.81469727e-06, -9.53674316e-07,  1.66786301e+00,
 1.51930523e+00,  1.38732100e+00,  1.44286168e+00,  1.45305979e+00,
 1.48680508e+00,  1.35569513e+00,  1.30651486e+00,  1.46690059e+00,
 1.46162009e+00,  1.45247817e+00,  1.50233293e+00,  1.58102441e+00,
 1.55677521e+00,  1.37420809e+00,  1.27815318e+00,  1.19939661e+00,
 1.20670402e+00,  1.18364000e+00,  1.20157003e+00,  1.16671526e+00,
 1.31985700e+00,  1.21735215e+00,  9.05602813e-01,  1.21752524e+00,
 1.16036272e+00,  1.42251587e+00,  1.33668280e+00,  1.25848365e+00,
 9.34405923e-01,  9.39077854e-01,  1.35263872e+00,  1.24485099e+00,
 0.00000000e+00,  2.79470325e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  5.08741379e+00,  5.40640831e+00,  5.49911833e+00,
 5.47265720e+00,  5.59636211e+00,  5.63628387e+00,  5.38047409e+00,
 5.48932981e+00,  0.00000000e+00]), array([ 1.48554025e-02, -2.47797254e-03, -7.42366351e-03,  9.21044964e-03,
-9.46046226e-03, -4.57502566e-02,  1.92341972e-02, -7.94130936e-02,
-5.84743917e-04,  6.02890968e+00,  6.11079597e+00,  6.09929609e+00,
 6.00685072e+00,  6.06542110e+00,  6.01126003e+00,  6.01112652e+00,
-1.57833099e-04, -5.46276569e-05,  1.97887421e-05, -9.38773155e-05,
 3.52859497e-05,  1.90734863e-05, -4.05907631e-05,  2.19345093e-05,
-5.36441803e-06, -5.24520874e-06,  2.86102295e-06,  7.39097595e-06,
-1.90734863e-06,  1.34706497e-05,  3.40938568e-05,  1.43051147e-06,
-4.76837158e-06, -5.72204590e-06,  1.90734863e-06,  1.57548046e+00,
 1.51535058e+00,  1.31750870e+00,  1.36473989e+00,  1.36661303e+00,
 1.40712237e+00,  1.31278157e+00,  1.30275285e+00,  1.47269487e+00,
 1.47635472e+00,  1.44900250e+00,  1.54277635e+00,  1.54253578e+00,
 1.57176065e+00,  1.31789446e+00,  1.31402183e+00,  1.26867115e+00,
 1.26883543e+00,  1.26307666e+00,  1.23084068e+00,  1.21563947e+00,
 1.34541035e+00,  1.28982627e+00,  9.85765994e-01,  9.68558908e-01,
 1.05537021e+00,  1.13667870e+00,  1.11146736e+00,  1.16311240e+00,
 7.05795407e-01,  1.01008010e+00,  1.27881575e+00,  1.17724717e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  4.94186068e+00,  5.27315807e+00,  5.34491253e+00,
 5.27993011e+00,  5.35590649e+00,  5.38039589e+00,  5.42045689e+00,
 5.33811808e+00,  0.00000000e+00]), array([ 9.62509401e-03,  4.62172460e-03, -7.83898681e-03,  2.24512629e-03,
-2.47435533e-02, -2.38912404e-02,  1.01141185e-02, -5.18663451e-02,
 1.51115656e-03,  6.29829311e+00,  6.35406256e+00,  6.33343315e+00,
 6.24153423e+00,  6.30987453e+00,  6.26516294e+00,  6.26847172e+00,
 5.12629747e-04, -6.07967377e-05, -2.31981277e-04,  1.17659569e-04,
 1.41233206e-04,  2.38090754e-04, -1.89870596e-04, -1.45435333e-04,
-2.56299973e-05, -1.03712082e-04,  3.64780426e-05,  1.94907188e-05,
-1.21593475e-05,  2.44379044e-05,  9.77516174e-06, -7.86781311e-06,
 7.86781311e-06,  1.43051147e-06,  2.86102295e-06,  1.90583324e+00,
 1.57333255e+00,  1.34851170e+00,  1.38652599e+00,  1.43546975e+00,
 1.45068192e+00,  1.33535993e+00,  1.35066223e+00,  1.46495211e+00,
 1.52903712e+00,  1.46730888e+00,  1.38579452e+00,  1.43440509e+00,
 1.36191607e+00,  1.26267374e+00,  1.24445033e+00,  1.16172218e+00,
 1.15860379e+00,  1.12153792e+00,  1.19810915e+00,  1.44587636e+00,
 1.60524917e+00,  1.55921984e+00,  1.29285431e+00,  1.18058348e+00,
 1.27835655e+00,  1.38483918e+00,  1.40247321e+00,  1.18130541e+00,
```

```
 6.78222716e-01,  9.21494484e-01,  1.25863624e+00,  1.31584167e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  5.14983892e+00,  5.51805782e+00,  5.57024813e+00,
 5.63419056e+00,  5.70388031e+00,  5.40296936e+00,  5.50691605e+00,
 5.60041332e+00,  0.00000000e+00]), array([ 3.63326957e-03,  4.64690244e-03, -1.01242941e-02,  1.04760937e-03,
-7.43335113e-03, -3.25428732e-02,  2.59018019e-02, -2.94070840e-02,
-1.11490488e-04,  7.43947411e+00,  7.48181963e+00,  7.49884987e+00,
 7.44430399e+00,  7.46032906e+00,  7.44401932e+00,  7.43931389e+00,
 1.70201063e-04, -9.19550657e-05, -2.16066837e-05,  6.52074814e-05,
 1.13129616e-04, -8.95857811e-05,  4.78029251e-05,  2.52127647e-05,
 6.92605972e-05, -1.31130219e-05,  1.18732452e-04, -1.95503235e-05,
 1.14440918e-04, -3.00407410e-05,  5.72204590e-06,  9.72747803e-05,
 1.23977661e-05, -1.33514404e-05,  4.05311584e-05,  1.98092675e+00,
 1.81861484e+00,  1.64888406e+00,  1.62645602e+00,  1.69311666e+00,
 1.70538247e+00,  1.61161602e+00,  1.64811933e+00,  1.82120252e+00,
 1.83177686e+00,  1.81338763e+00,  1.84273791e+00,  1.76277399e+00,
 1.80616641e+00,  1.54324245e+00,  1.48999727e+00,  1.49985480e+00,
 1.45427072e+00,  1.43338013e+00,  1.43896604e+00,  1.37414002e+00,
 1.52426434e+00,  1.52030420e+00,  1.25127387e+00,  1.55662775e+00,
 1.68743122e+00,  1.70097244e+00,  1.64307463e+00,  1.41273201e+00,
 9.46535587e-01,  1.18759108e+00,  1.43940043e+00,  1.49260020e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  6.17862654e+00,  6.49424267e+00,  6.52763748e+00,
 6.57387161e+00,  6.66408348e+00,  6.70301628e+00,  6.43536282e+00,
 6.63190651e+00,  0.00000000e+00]), array([ 7.50044035e-03,  2.11319793e-03,  6.06318004e-04,  2.29338370e-03,
-1.42199546e-03, -1.95934959e-02,  9.87889990e-03, -6.85371608e-02,
-3.14012170e-04,  2.15790987e+00,  2.18920159e+00,  2.22633433e+00,
 2.16394615e+00,  2.20072126e+00,  2.15806508e+00,  2.18500471e+00,
-1.66893005e-06, -2.97725201e-05, -3.18288803e-05, -1.37388706e-05,
 1.09672546e-05,  1.83582306e-05,  1.53183937e-05, -1.15931034e-05,
 4.22894955e-05, -4.35113907e-06,  4.17232513e-06,  1.06096268e-05,
 8.40425491e-06, -4.58955765e-06, -7.03334808e-06, -6.25848770e-06,
-2.38418579e-07, -2.26497650e-06,  1.54972076e-06,  7.32911587e-01,
 5.14674485e-01,  4.09921765e-01,  4.41726327e-01,  5.32732964e-01,
 4.48109657e-01,  3.71026218e-01,  4.18896139e-01,  6.01859570e-01,
 6.06205821e-01,  6.12362146e-01,  7.40438819e-01,  7.02321649e-01,
 8.02494287e-01,  5.09593248e-01,  5.07661045e-01,  5.35103440e-01,
 4.83254611e-01,  4.66021717e-01,  5.15228748e-01,  5.73815823e-01,
 6.10063076e-01,  5.98458111e-01,  3.64616007e-01,  3.29670906e-01,
 4.05514002e-01,  2.81637788e-01,  4.20609534e-01,  2.88983881e-01,
-2.06432700e-01,  7.95636177e-02,  2.96474516e-01,  2.88832963e-01,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  1.71287894e+00,  1.97777593e+00,  1.99797750e+00,
 1.89500332e+00,  1.89797831e+00,  1.87078309e+00,  1.91662943e+00,
 2.01164484e+00,  0.00000000e+00]), array([-2.59139785e-03,  6.43054582e-03, -6.09295815e-03,  2.08649412e-03,
-2.53454298e-02,  4.32593375e-03, -2.43431330e-03, -6.76562712e-02,
 1.00687146e-04,  5.60367155e+00,  5.63616753e+00,  5.62022114e+00,
 5.60126686e+00,  5.58994722e+00,  5.57238913e+00,  5.60529995e+00,
 1.24499202e-04,  2.66432762e-05,  1.60932541e-06, -4.17232513e-06,
-1.30534172e-05, -5.77569008e-05,  1.82390213e-05,  1.01327896e-06,
-7.86781311e-06, -9.53674316e-07,  4.76837158e-06, -5.24520874e-06,
 0.00000000e+00,  3.81469727e-06, -5.24520874e-06,  1.43051147e-06,
 9.53674316e-07, -9.53674316e-07, -9.53674316e-07,  1.57843828e+00,
 1.31552577e+00,  1.23192537e+00,  1.26475310e+00,  1.33404982e+00,
 1.22843742e+00,  1.18432045e+00,  1.22492230e+00,  1.40769482e+00,
 1.34247541e+00,  1.38500834e+00,  1.40429568e+00,  1.35370469e+00,
 1.37643766e+00,  1.15103590e+00,  1.11030960e+00,  1.09689069e+00,
 1.07472372e+00,  1.04447436e+00,  1.10478830e+00,  1.12130773e+00,
 1.19993389e+00,  1.06334269e+00,  9.67672706e-01,  1.12304783e+00,
 1.12455213e+00,  1.00748110e+00,  1.08210957e+00,  1.13148975e+00,
 7.88510799e-01,  9.60384011e-01,  1.18535423e+00,  1.26765132e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  4.60339642e+00,  4.91069937e+00,  4.97803831e+00,
 4.99830103e+00,  5.04726267e+00,  5.02607155e+00,  4.83888197e+00,
 4.83525705e+00,  0.00000000e+00]), array([ 1.05654192e-03,  8.10640678e-03, -5.60208410e-03,  3.20837274e-03,
-1.92926750e-02,  8.40626657e-03, -6.87092543e-04, -5.68793043e-02,
 7.47963786e-05,  5.31474209e+00,  5.33405209e+00,  5.26405191e+00,
 5.30303812e+00,  5.28858328e+00,  5.25658178e+00,  5.26834583e+00,
 1.36613846e-04,  3.09348106e-05,  3.21269035e-05,  5.10215759e-05,
-3.37362289e-05,  9.54866409e-05, -3.01003456e-05, -3.04877758e-05,
 3.09944153e-06,  9.41753387e-06, -2.25305557e-05, -8.85128975e-05,
-8.82148743e-06,  7.98702240e-06, -2.52723694e-05, -2.64644623e-05,
 1.66893005e-05, -5.00679016e-06, -1.90734863e-06,  1.58908427e+00,
 1.28850341e+00,  1.14269924e+00,  1.16208684e+00,  1.29685497e+00,
 1.19950914e+00,  1.10114419e+00,  1.15449524e+00,  1.27103186e+00,
 1.21286869e+00,  1.23110271e+00,  1.35316920e+00,  1.37750769e+00,
 1.41206419e+00,  1.17332244e+00,  1.18677652e+00,  1.18676341e+00,
 1.16448593e+00,  1.15114260e+00,  1.22649944e+00,  1.11902308e+00,
 1.25072920e+00,  1.06343675e+00,  1.03759587e+00,  9.25293684e-01,
 9.00614738e-01,  7.95112729e-01,  9.84687805e-01,  8.90600920e-01,
 3.96062136e-01,  7.10371733e-01,  1.02083087e+00,  1.05244374e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  4.27194357e+00,  4.65103579e+00,  4.72581196e+00,
 4.61160374e+00,  4.60442972e+00,  4.66168404e+00,  4.75528955e+00,
 4.74599648e+00,  0.00000000e+00])]
```

Расчет качества

Ввод [13]:
```
def calculate_model (x):
    lr = -1
    model = energy_lr[x.building_id][x.hour]
    if len(model) > 0:
        lr = np.sum([x[col] * model[i] for i,col in enumerate(lr_columns[3:])])
        lr += model[len(lr_columns)-3]
        lr = np.exp(lr)
    if lr < 0:
        lr = 0
    x["meter_reading_lr_q"] = (np.log(x.meter_reading + 1) -
                               np.log(1 + lr))**2
    return x

energy_test = energy_test.apply(calculate_model,
                                axis=1, result_type="expand")
energy_test_lr_rmsle = np.sqrt(energy_test["meter_reading_lr_q"].sum() / len(energy_test))
print ("Качество линейной репрессии, 20 зданий:",
       energy_test_lr_rmsle, round(energy_test_lr_rmsle, 1))
```

```
Качество линейной регрессии, 20 зданий: 0.20674574380055222 0.2
```

ИЕРАРХИЯ МОДЕЛЕЙ

Постановка задачи

Посчитаем модель линейной регрессии по первым 100 зданиям и найдем ее точность, используя в качестве параметров только дни недели и праздники, применяя fit_intercept=False и логарифмируя целевой показатель.

Для вычисления отсутствующих или некорректных данных построим модели по всем зданиям одного типа в одном городе и во всех городах.

Данные:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz
2. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
import numpy as np
from scipy.interpolate import interp1d
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

Загрузка данных, отсечение 100 зданий, объединение и оптимизация

Ввод [2]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
```

```
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        elif col == "timestamp":
            df[col] = pd.to_datetime(df[col])
        elif str(col_type)[:8] != "datetime":
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                            / start_mem, 1), '%)')
    return df
```

Ввод [3]:
```
buildings = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz")
weather = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz")
energy = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz")
energy = energy[(energy["building_id"]<100)]
energy = pd.merge(left=energy, right=buildings, how="left",
                  left_on="building_id", right_on="building_id")
energy = energy.set_index(["timestamp", "site_id"])
weather = weather.set_index(["timestamp", "site_id"])
energy = pd.merge(left=energy, right=weather, how="left",
                  left_index=True, right_index=True)
energy.reset_index(inplace=True)
energy = energy.drop(columns=["meter", "year_built",
                              "square_feet", "floor_count"], axis=1)
del buildings
del weather
energy = reduce_mem_usage(energy)
print (energy.info())
```

```
Потребление памяти меньше на 56.89 Мб (минус 71.9 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 864557 entries, 0 to 864556
Data columns (total 12 columns):
timestamp             864557 non-null datetime64[ns]
site_id               864557 non-null int8
building_id           864557 non-null int8
meter_reading         864557 non-null float16
primary_use           864557 non-null category
air_temperature       864263 non-null float16
cloud_coverage        487693 non-null float16
dew_temperature       864263 non-null float16
precip_depth_1_hr     864459 non-null float16
sea_level_pressure    856210 non-null float16
wind_direction        839970 non-null float16
wind_speed            864557 non-null float16
dtypes: category(1), datetime64[ns](1), float16(8), int8(2)
memory usage: 22.3 MB
None
```

Обогащение данных: час, дни недели, праздники, логарифм

Ввод [11]:
```
energy["hour"] = energy["timestamp"].dt.hour.astype("int8")
energy["weekday"] = energy["timestamp"].dt.weekday.astype("int8")
for weekday in range(0,7):
    energy['is_wday' + str(weekday)] = energy['weekday'].isin([weekday]).astype("int8")
energy["date"] = pd.to_datetime(energy["timestamp"].dt.date)
dates_range = pd.date_range(start='2015-12-31', end='2017-01-01')
us_holidays = calendar().holidays(start=dates_range.min(),
                                  end=dates_range.max())
energy['is_holiday'] = energy['date'].isin(us_holidays).astype("int8")
energy["meter_reading_log"] = np.log(energy["meter_reading"] + 1)
```

Разделение данных

Ввод [13]:
```
energy_train, energy_test = train_test_split(energy[(energy["meter_reading"]>0)], test_size=0.2)
print (energy_train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 427872 entries, 860794 to 336212
Data columns (total 24 columns):
timestamp             427872 non-null datetime64[ns]
site_id               427872 non-null int8
building_id           427872 non-null int8
```

```
meter_reading         427872 non-null float16
primary_use           427872 non-null category
air_temperature       427867 non-null float16
cloud_coverage        248978 non-null float16
dew_temperature       427867 non-null float16
precip_depth_1_hr     427870 non-null float16
sea_level_pressure    425639 non-null float16
wind_direction        414132 non-null float16
wind_speed            427872 non-null float16
hour                  427872 non-null int8
weekday               427872 non-null int8
is_wday0              427872 non-null int8
is_wday1              427872 non-null int8
is_wday2              427872 non-null int8
is_wday3              427872 non-null int8
is_wday4              427872 non-null int8
is_wday5              427872 non-null int8
is_wday6              427872 non-null int8
date                  427872 non-null datetime64[ns]
is_holiday            427872 non-null int8
meter_reading_log     427872 non-null float16
dtypes: category(1), datetime64[ns](2), float16(9), int8(12)
memory usage: 22.4 MB
None
```

Линейная регрессия: по часам

Ввод [15]:
```
hours = range(0, 24)
buildings = range(0, energy_train["building_id"].max() + 1)
lr_columns = ["meter_reading_log", "hour", "building_id", "is_holiday"]
for wday in range(0,7):
    lr_columns.append("is_wday" + str(wday))
energy_train_lr = pd.DataFrame(energy_train, columns=lr_columns)
energy_lr = [[]]*len(buildings)
for building in buildings:
    energy_lr[building] = [[]]*len(hours)
    energy_train_b = energy_train_lr[energy_train_lr["building_id"]==building]
    for hour in hours:
        energy_lr[building].append([0]*(len(lr_columns)-3))
        energy_train_bh = pd.DataFrame(energy_train_b[energy_train_b["hour"]==hour])
        y = energy_train_bh["meter_reading_log"]
        if len(y) > 0:
            x = energy_train_bh.drop(labels=["meter_reading_log",
                "hour", "building_id"], axis=1)
            model = LinearRegression(fit_intercept=False).fit(x, y)
            energy_lr[building][hour] = model.coef_
            energy_lr[building][hour] = np.append(energy_lr[building][hour], model.intercept_)
print (energy_lr[0])
```

```
[array([-0.07672264,  5.44733172,  5.48182745,  5.47721354,  5.42741865,
        5.48529313,  5.48270089,  5.48386549,  0.        ]), array([-0.15666387,  5.4587836 ,  5.45023148,  5.45401278,  5.
48535156,
        5.45342968,  5.47435462,  5.47386853,  0.        ]), array([-0.11096011,  5.43725553,  5.42357337,  5.4938151 ,  5.
46340231,
        5.46114871,  5.44015625,  5.42169744,  0.        ]), array([-0.15330244,  5.45772624,  5.46831597,  5.49652778,  5.
46149029,
        5.4555444 ,  5.48141164,  5.44215745,  0.        ]), array([-0.13053489,  5.46305359,  5.41878255,  5.48370151,  5.
44810116,
        5.45115642,  5.44285301,  5.42983774,  0.        ]), array([-0.13158166,  5.46870578,  5.45918642,  5.47578125,  5.
45368304,
        5.4471477 ,  5.43457031,  5.42787388,  0.        ]), array([-0.18098801,  5.5090577 ,  5.40987723,  5.46679688,  5.
45813295,
        5.45484555,  5.41682943,  5.45173891,  0.        ]), array([-0.11250546,  5.49010508,  5.47352431,  5.49881114,  5.
49606604,
        5.45292143,  5.44463315,  5.44677734,  0.        ]), array([-0.03939952,  5.48428679,  5.47381366,  5.50441576,  5.
4825067 ,
        5.4790625 ,  5.37247721,  5.45587384,  0.        ]), array([-0.09700478,  5.41360953,  5.44010417,  5.48703125,  5.
42906148,
        5.49071661,  5.43894676,  5.43628772,  0.        ]), array([-0.1309545 ,  5.50148183,  5.45968192,  5.45061384,  5.
45476943,
        5.41153419,  5.38421875,  5.38850911,  0.        ]), array([-0.11626676,  5.4272554 ,  5.40925481,  5.46839489,  5.
44738045,
        5.43572979,  5.37282986,  5.40609375,  0.        ]), array([-0.13854069,  5.50320298,  5.40828125,  5.46234375,  5.
43659935,
        5.41446394,  5.35716146,  5.42047991,  0.        ]), array([-0.07534867,  5.43519061,  5.4461263 ,  5.45341435,  5.
4362952 ,
        5.49875   ,  5.41294643,  5.41822917,  0.        ]), array([-0.08523259,  5.42027178,  5.47214674,  5.49609375,  5.
42545366,
        5.35887805,  5.42623197,  5.39990234,  0.        ]), array([-0.14730212,  5.49009646,  5.48486328,  5.47332974,  5.
40260056,
        5.43975072,  5.42082332,  5.4390625 ,  0.        ]), array([-0.11099745,  5.43354288,  5.47893415,  5.51494565,  5.
42524269,
        5.42063301,  5.37328125,  5.38392857,  0.        ]), array([-0.17338942,  5.50514663,  5.47115385,  5.53296875,  5.
44644668,
        5.40951486,  5.39894701,  5.36422631,  0.        ]), array([-0.13046875,  5.4234375 ,  5.4695638 ,  5.48863636,  5.
46525493,
        5.4447338 ,  5.45136719,  5.42464193,  0.        ]), array([-0.09309772,  5.43587216,  5.45661272,  5.47144397,  5.
47388016,
        5.43984936,  5.42083333,  5.42751736,  0.        ]), array([-0.07530817,  5.40191039,  5.4565625 ,  5.48336227,  5.
```

```
48374302,
        5.44921875,  5.47413793,  5.44612069,  0.        ]), array([-0.05997873,  5.41457605,  5.47816051,  5.48524306,  5.
42287101,
        5.45447838,  5.46108774,  5.42459239,  0.        ]), array([-0.18531843,  5.5223072 ,  5.47723599,  5.48200335,  5.
44227391,
        5.45931412,  5.42626953,  5.43518519,  0.        ]), array([-0.1275144 ,  5.4619648 ,  5.47686298,  5.43012153,  5.
44713183,
        5.42712663,  5.4609375 ,  5.46440972,  0.        ]), [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0,
0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0],
0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0,
0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0]]
```

Линейная регрессия: по типам зданий

Ввод [16]:
```
sites = range(0, energy["site_id"].max() + 1)
primary_uses = energy["primary_use"].unique()
lr_columns_use = ["meter_reading_log", "hour", "building_id",
                  "is_holiday", "primary_use", "site_id"]
for wday in range(0,7):
    lr_columns_use.append("is_wday" + str(wday))
energy_lr_use = {}
energy_lr_use_site = {}
energy_train_lr = pd.DataFrame(energy_train, columns=lr_columns_use)
for primary_use in primary_uses:
    energy_train_u = energy_train_lr[energy_train_lr["primary_use"]==primary_use]
    if len(energy_train_u) > 0:
        energy_lr_use_site[primary_use] = [[]]*len(sites)
        for site in sites:
            energy_lr_use_site[primary_use][site] = [[]]*len(hours)
            energy_train_us = energy_train_u[energy_train_u["site_id"]==site]
            if len(energy_train_us) > 0:
                for hour in hours:
                    energy_train_uth = energy_train_us[energy_train_us["hour"]==hour]
                    y = energy_train_uth["meter_reading_log"]
                    if len(y) > 0:
                        x = energy_train_uth.drop(labels=["meter_reading_log",
                            "hour", "building_id", "site_id",
                            "primary_use"], axis=1)
                        model = LinearRegression(fit_intercept=False).fit(x, y)
                        energy_lr_use_site[primary_use][site][hour] = model.coef_
                        energy_lr_use_site[primary_use][site][hour] = np.append(energy_lr_use_site[primary_use]
                                                                                [site][hour], model.intercept_)
        energy_lr_use[primary_use] = [[]]*len(hours)
        for hour in hours:
            energy_train_th = energy_train_u[energy_train_u["hour"]==hour]
            y = energy_train_th["meter_reading_log"]
            if len(y) > 0:
                x = energy_train_th.drop(labels=["meter_reading_log",
                    "hour", "building_id", "site_id", "primary_use"], axis=1)
                model = LinearRegression(fit_intercept=False).fit(x, y)
                energy_lr_use[primary_use][hour] = model.coef_
                energy_lr_use[primary_use][hour] = np.append(energy_lr_use[primary_use][hour], model.intercept_)
print (energy_lr_use_site)
```

```
{'Education': [[array([-4.95108910e-03,  5.61360437e+00,  5.65882530e+00,  5.67235354e+00,
        5.68624227e+00,  5.66868724e+00,  5.66855781e+00,  5.61365798e+00,
        0.00000000e+00]), array([-0.09751058,  5.57509723,  5.63411768,  5.64540389,  5.67529104,
        5.64540242,  5.62053826,  5.5898385 ,  0.        ]), array([-0.09047305,  5.58844298,  5.59584691,  5.6420983
6,  5.62227457,
        5.64149599,  5.58717135,  5.55590392,  0.        ]), array([-0.07769471,  5.6086802 ,  5.63253348,  5.6176903
8,  5.64547642,
        5.6607634 ,  5.61264597,  5.58146314,  0.        ]), array([-0.11248114,  5.62500199,  5.6085948 ,  5.6301456
3,  5.65697546,
        5.63792365,  5.58827664,  5.56795446,  0.        ]), array([-0.00847801,  5.60303734,  5.6189954 ,  5.6188826
4,  5.67267042,
        5.62066229,  5.64663652,  5.5692787 ,  0.        ]), array([-0.05374606,  5.73390029,  5.71455242,  5.7009349
3,  5.70687447,
        5.71191509,  5.68832156,  5.7302437 ,  0.        ]), array([-0.13240948,  5.82400028,  5.8334283 ,  5.8228120
7,  5.827397  ,
        5.83145864,  5.77359089,  5.78109099,  0.        ]), array([0.06579212, 5.83295367, 5.89667629, 5.91281993,
5.87555637,
        5.90514838, 5.79579004, 5.78730125, 0.        ]), array([-0.12028337,  5.95981853,  5.92581263,  5.93303184,
5.93486272,
        5.91308583,  5.82147555,  5.79067072,  0.        ]), array([-0.16587912,  5.99228824,  6.00498112,  5.9439272
8,  5.97863522,
        5.96636872,  5.81502467,  5.83363012,  0.        ]), array([-0.23159424,  6.06743021,  5.98602393,  5.9933460
2,  5.99975008,
        6.02667127,  5.75990739,  5.82637219,  0.        ]), array([-0.12871637,  6.02775242,  6.00720321,  6.0452657
5,  6.04163365,
        5.98119665,  5.82243924,  5.84899643,  0.        ]), array([-0.12243417,  6.04303105,  6.04584842,  6.0371372
,  6.0420403 ,
        6.00815957,  5.85007128,  5.84130598,  0.        ]), array([-0.1972073 ,  6.05708315,  6.11829698,  6.0615394
7,  6.03556397,
        5.97645389,  5.82135485,  5.81270059,  0.        ]), array([-0.12170295,  6.04465485,  6.05629214,  6.0292924
7,  6.03582924,
        5.97293836,  5.86171643,  5.82888345,  0.        ]), array([-0.19968386,  6.05249116,  6.04567473,  6.0031001
1,  6.0106921 ,
        5.9659547 ,  5.81804076,  5.86894322,  0.        ]), array([-0.19222995,  5.9376534 ,  5.99224657,  5.9779083
```

```
, 5.97782127,
       5.93488909, 5.85915911, 5.84523737, 0.        ]), array([-0.08678401, 5.96689775, 5.9628178 , 5.9854516
5, 5.97211423,
       5.89372649, 5.87871094, 5.84288996, 0.        ]), array([-0.09793399, 5.90877736, 5.92011406, 5.9170767
, 5.91080001,
       5.84582395, 5.83202188, 5.86175903, 0.        ]), array([0.03306855, 5.89769254, 5.89317047, 5.88910091,
5.9599016 ,
       5.85809921, 5.84204777, 5.82314479, 0.        ]), array([-0.03340297, 5.89308984, 5.90217284, 5.91360844,
5.90272226,
       5.88582599, 5.85517356, 5.8079545 , 0.        ]), array([-0.10623371, 5.86244938, 5.84447788, 5.8172225
6, 5.83867277,
       5.84216423, 5.80268113, 5.80461822, 0.        ]), array([-0.0563033 , 5.69444707, 5.71601056, 5.7056417
7, 5.73093134,
       5.67392559, 5.62333997, 5.66524352, 0.        ])]], 'Lodging/residential': [[array([-0.04523372, 5.608456
64, 5.58814653, 5.58628753, 5.59600761,
       5.5536678 , 5.56357219, 5.5621789 , 0.        ]), array([-0.02834239, 5.5692455 , 5.5492541 , 5.5480979
6, 5.53853915,
       5.52052036, 5.50672522, 5.53950517, 0.        ]), array([-2.10616534e-03, 5.49254322e+00, 5.49772326e+0
0, 5.51181203e+00,
       5.50521155e+00, 5.49562673e+00, 5.51151748e+00, 5.49647022e+00,
       0.00000000e+00]), array([-0.05516948, 5.48655037, 5.49665601, 5.46550871, 5.4901299 ,
       5.46841561, 5.48563895, 5.51463105, 0.        ]), array([-1.52576416e-04, 5.45188761e+00, 5.46641425e+0
0, 5.48171579e+00,
       5.46816163e+00, 5.45094582e+00, 5.47295271e+00, 5.46690781e+00,
       0.00000000e+00]), array([-0.04199569, 5.44638205, 5.4533773 , 5.44055268, 5.43449295,
       5.45143388, 5.45494528, 5.44695337, 0.        ]), array([-0.04463606, 5.46408156, 5.45508687, 5.4498075
6, 5.44155541,
       5.43212821, 5.43878774, 5.43902169, 0.        ]), array([-0.06023721, 5.43295514, 5.42802978, 5.4291127
1, 5.4194954 ,
       5.40537115, 5.3943706 , 5.40270177, 0.        ]), array([-0.07452139, 5.45562758, 5.43664239, 5.4432521
6, 5.42581218,
       5.44006535, 5.38857309, 5.40767283, 0.        ]), array([-0.09551625, 5.49098315, 5.45978979, 5.4864212
7, 5.46381898,
       5.45141862, 5.41232544, 5.43115883, 0.        ]), array([-0.03525881, 5.50296517, 5.51886853, 5.5070800
8, 5.49906725,
       5.48423844, 5.43710537, 5.45375047, 0.        ]), array([-0.06151838, 5.54606927, 5.52348156, 5.5030502
9, 5.51776884,
       5.50610813, 5.45627124, 5.49212182, 0.        ]), array([-0.04750607, 5.56633248, 5.55592045, 5.5535316
3, 5.54609251,
       5.54976197, 5.51154274, 5.53081565, 0.        ]), array([-0.05126138, 5.58046838, 5.56430964, 5.5622702
2, 5.55392528,
       5.53837577, 5.52867724, 5.55224754, 0.        ]), array([-0.10030835, 5.58837975, 5.58343672, 5.5582643
4, 5.57725133,
       5.56117577, 5.55578301, 5.57918847, 0.        ]), array([-0.02493995, 5.60232181, 5.58821467, 5.5676901
6, 5.58093861,
       5.57042014, 5.55251116, 5.59891596, 0.        ]), array([-0.04887183, 5.60770543, 5.58964291, 5.5790898
7, 5.58209557,
       5.59207353, 5.56107305, 5.59671336, 0.        ]), array([-0.05707969, 5.61945682, 5.59873776, 5.5992638
2, 5.59667723,
       5.59563006, 5.57593272, 5.57307799, 0.        ]), array([-0.03568851, 5.6172074 , 5.62923476, 5.6228550
5, 5.61446774,
       5.5769749 , 5.57312558, 5.60773593, 0.        ]), array([-0.0838263 , 5.66052009, 5.62732817, 5.6320453
1, 5.61495727,
       5.61393458, 5.59155101, 5.6633669 , 0.        ]), array([-0.0923647 , 5.66423455, 5.63566332, 5.6325712
7, 5.62879631,
       5.60020285, 5.603958  , 5.6594769 , 0.        ]), array([-0.04457152, 5.6397771 , 5.64038948, 5.6357092
1, 5.61316252,
       5.58956338, 5.56984135, 5.64606004, 0.        ]), array([-0.09074348, 5.65755347, 5.61949655, 5.6290422
5, 5.58670905,
       5.58824353, 5.56988191, 5.64498377, 0.        ]), array([-0.0582248 , 5.63232509, 5.59979525, 5.6335074
8, 5.60156387,
       5.5703105 , 5.56575037, 5.61623539, 0.        ])]], 'Office': [[array([-0.10589314, 4.98789043, 4.935823
23, 4.91372822, 4.96732076,
       4.91446987, 4.94595359, 4.93767619, 0.        ]), array([-0.05104407, 4.96262271, 4.93994828, 4.9195780
9, 4.91061053,
       4.93051172, 4.9120206 , 4.94962054, 0.        ]), array([0.0195464 , 4.99615335, 4.91905167, 4.91090544,
4.89669365,
       4.96963901, 4.91173232, 4.96716597, 0.        ]), array([-0.02352759, 4.97528208, 4.96828957, 4.96587112,
4.93527768,
       4.92759609, 4.91235214, 4.94535072, 0.        ]), array([-0.04912426, 4.99814276, 4.92725082, 4.9336754
5, 4.97252992,
       4.98535128, 4.91359314, 4.93548271, 0.        ]), array([0.043165  , 5.03639046, 4.97928954, 4.97126389,
5.00971253,
       4.9568101 , 4.95006654, 4.96925428, 0.        ]), array([-0.10533133, 5.09276098, 5.09775681, 5.09512755,
5.10154437,
       5.09229378, 5.05072846, 5.01509127, 0.        ]), array([-0.09799276, 5.20097188, 5.15230507, 5.1452107
6, 5.12381112,
       5.16564764, 5.040548  , 4.96191978, 0.        ]), array([-0.28143476, 5.25424927, 5.2336816 , 5.2582950
8, 5.2228226 ,
       5.21597161, 5.00385669, 4.92148817, 0.        ]), array([-0.24176715, 5.2668436 , 5.30337536, 5.2588702
9, 5.26761129,
       5.22338826, 5.01684845, 4.94548788, 0.        ]), array([-0.204122  , 5.3030783 , 5.28479946, 5.2783300
6, 5.29313775,
       5.26037267, 4.99180537, 4.95077819, 0.        ]), array([-0.23597543, 5.31881227, 5.30723904, 5.2526781
7, 5.27954674,
       5.32492619, 5.00067637, 4.95629691, 0.        ]), array([-0.26219442, 5.34233976, 5.35650502, 5.3022860
5, 5.30100494,
       5.29220473, 5.01310082, 4.98918443, 0.        ]), array([-0.24302324, 5.32705602, 5.3450472 , 5.2730705
7, 5.30999015,
       5.25498463, 5.04222842, 4.9699871 , 0.        ]), array([-0.25136347, 5.29820984, 5.33376539, 5.3158110
1, 5.28566734,
       5.23509378, 5.01789202, 4.99538352, 0.        ]), array([-0.27716022, 5.33731961, 5.31831473, 5.3405543
```

```
2,  5.32236274,
        5.27859443,  5.02494303,  4.95856531,  0.        ]), array([-0.24689704,  5.30619947,  5.31359111,  5.2908963
5,  5.29877122,
        5.25695126,  4.99370155,  5.01011742,  0.        ]), array([-0.13418939,  5.21239909,  5.20267231,  5.2229183
9,  5.25073511,
        5.20489773,  5.01515459,  4.9978731 ,  0.        ]), array([-0.0716772 ,  5.13514414,  5.18100068,  5.1648370
2,  5.22103786,
        5.15412908,  5.05669291,  5.04187319,  0.        ]), array([-0.08222602,  5.15012294,  5.15238571,  5.1501941
6,  5.14782385,
        5.16108956,  5.0844184 ,  5.0332404 ,  0.        ]), array([0.02724748, 5.09427126, 5.15043123, 5.12488151,
5.12654783,
        5.11744194, 5.02317541, 5.03373266, 0.        ]), array([-0.09096868,  5.08055502,  5.10104317,  5.10491555,
5.07425248,
        5.10066471,  5.01852689,  5.02482175,  0.        ]), array([-0.12719747,  5.02601647,  5.01818801,  5.0235948
2,  5.02146416,
        4.97576789,  4.99300728,  5.00633808,  0.        ]), array([-0.1571731 ,  4.96436644,  4.95503524,  4.9280208
3,  4.92043778,
        4.88405546,  4.93613916,  4.93222284,  0.        ])]], 'Entertainment/public assembly': [[array([0.09181222,
5.30748577, 5.34860992, 5.37867338, 5.32757695,
        5.21187273, 5.33217508, 5.37707713, 0.        ]), array([0.08528475, 5.1757638 , 5.2833252 , 5.31836713, 5.182
80634,
        5.3218157 , 5.13451385, 5.25413217, 0.        ]), array([-0.17911851,  5.2751693 ,  5.23290241,  5.27318618,
5.20636076,
        5.11171601,  5.26128906,  5.26468629,  0.        ]), array([0.08161648, 5.23082465, 5.322082  , 5.1733871 ,
5.16304931,
        5.32379789, 5.19227013, 5.18437349, 0.        ]), array([0.16350698, 5.13666177, 5.20305791, 5.20645591, 5.268
56457,
        5.24128618, 5.24361256, 5.16604791, 0.        ]), array([-0.26673858,  5.45032394,  5.3381235 ,  5.2500775 ,
5.33435209,
        5.21430374,  5.21044159,  5.17368966,  0.        ]), array([-0.03066297,  5.61077945,  5.63942005,  5.5458688
4,  5.42616872,
        5.50948679,  5.42213007,  5.38386249,  0.        ]), array([-0.04669814,  5.83427856,  5.65187027,  5.6080594
7,  5.64702678,
        5.69395221,  5.49188368,  5.34287029,  0.        ]), array([-0.64168152,  5.89256382,  5.87253808,  5.7014322
9,  5.77590623,
        5.84825121,  5.58429046,  5.4436452 ,  0.        ]), array([-0.33135681,  5.97376898,  6.0296534 ,  5.8087968
7,  5.95336382,
        5.81298918,  5.55452112,  5.62102171,  0.        ]), array([-0.32579558,  6.04026796,  5.90870667,  5.9870485
4,  5.87940936,
        5.92326705,  5.72085905,  5.54596094,  0.        ]), array([-0.45515126,  6.06263178,  6.08255413,  5.9418989
3,  5.97946522,
        5.88810755,  5.75573291,  5.51125872,  0.        ]), array([-0.09704072,  6.03663716,  6.04960029,  5.8952694
4,  5.83637585,
        5.82050185,  5.75694776,  5.59884644,  0.        ]), array([-0.3877708 ,  5.98430775,  5.87826563,  5.9768979
6,  5.82447614,
        5.79896673,  5.62677144,  5.64687803,  0.        ]), array([-0.11920715,  5.95059236,  6.03224981,  5.8304656
5,  5.85183295,
        5.83513006,  5.7812283 ,  5.67613181,  0.        ]), array([-0.14898554,  5.99032987,  6.00278793,  5.8753305
3,  5.92244859,
        5.87312416,  5.67829446,  5.75589305,  0.        ]), array([-0.18107695,  5.99568323,  5.93339224,  5.9956217
4,  5.87349789,
        5.92258186,  5.7073193 ,  5.6131429 ,  0.        ]), array([-0.32795974,  5.95009093,  5.87080608,  5.8408790
5,  5.82807242,
        5.82438401,  5.8447345 ,  5.60225528,  0.        ]), array([0.04867305, 5.99106144, 6.00769043, 5.90796686,
5.86605676,
        5.77397322, 5.78016745, 5.74366714, 0.        ]), array([-0.24379826,  5.97479619,  5.98090978,  5.95214037,
5.83876296,
        5.70259133,  5.70399558,  5.60092885,  0.        ]), array([-0.14094898,  5.93719694,  5.93201188,  6.0323638
9,  5.91205507,
        5.80599457,  5.81726792,  5.5064623 ,  0.        ]), array([-0.03474389,  5.76872043,  5.59588623,  5.7736080
1,  5.63615242,
        5.61407708,  5.60446912,  5.51633113,  0.        ]), array([-0.23055137,  5.64794683,  5.53668162,  5.5298614
5,  5.62938956,
        5.53917754,  5.48534393,  5.3824567 ,  0.        ]), array([-0.10794854,  5.31454305,  5.45888199,  5.4330234
4,  5.51142793,
        5.48873256,  5.39041303,  5.27396678,  0.        ])]], 'Other': [[array([-0.30719241,  4.14420546,  4.0016128
1,  4.02872532,  4.12182625,
        4.05322085,  3.9404807 ,  3.90168581,  0.        ]), array([-0.11271853,  4.0451028 ,  4.01367563,  4.0335108
7,  4.03461852,
        4.10183768,  4.06872093,  4.03279943,  0.        ]), array([0.03250139, 4.03642008, 3.89515653, 4.15750614,
4.02282895,
        3.95832645, 3.97694817, 3.92525391, 0.        ]), array([-0.08807834,  4.0076697 ,  3.94845581,  3.98406058,
4.08466696,
        3.90182241,  3.98680906,  3.96163448,  0.        ]), array([-0.00681196,  3.85962679,  4.07766677,  3.9620780
4,  3.96142855,
        3.98327638,  4.03070706,  4.03353377,  0.        ]), array([0.06945012, 3.920814  , 3.93799222, 3.98009581,
3.97828059,
        3.88881162, 3.95159428, 4.08577741, 0.        ]), array([-0.1023194 ,  3.92211508,  3.97380447,  4.02470779,
4.01562944,
        3.98785807,  4.11557839,  3.94545898,  0.        ]), array([0.13801113, 3.94145569, 3.89773537, 3.98494991,
3.98967487,
        3.84246255, 3.90014114, 3.87407978, 0.        ]), array([-0.02300967,  3.95617009,  3.9864064 ,  3.86758448,
3.89071876,
        3.73894555,  3.74360862,  3.75582531,  0.        ]), array([-0.16275012,  3.96259158,  3.87333761,  3.9547001
,  3.79775565,
        3.91126449,  3.66278631,  3.71430164,  0.        ]), array([0.08991502, 4.05740461, 3.87738328, 3.84679594,
3.83952737,
        4.05600072, 3.92495521, 3.78230323, 0.        ]), array([0.02388226, 3.91756297, 3.98574916, 3.85330859, 3.877
55879,
        3.80387674, 3.80037497, 3.88107374, 0.        ]), array([-0.06889435,  4.02263642,  4.15243479,  4.08902995,
3.82429212,
        3.92497757,  3.87077531,  4.03839158,  0.        ]), array([-0.11148173,  3.86129534,  3.91349348,  4.1415463
```

```
7,  4.07679955,
       3.92204483,  3.92809245,  4.06853623,  0.        ]), array([0.02898037, 3.9678678 , 3.98110634, 4.05909717,
4.00279294,
       3.88439333, 3.84833984, 4.07661057, 0.        ]), array([-0.31533817,  4.0804682 ,  4.13576562,  4.14784241,
4.05495447,
       4.01095061,  3.98444528,  4.04120572,  0.        ]), array([-0.16071049,  4.04183885,  4.01053703,  4.1066802
7,  4.12649092,
       4.11658118,  3.96772676,  4.02592106,  0.        ]), array([-0.27895654,  4.10018533,  4.06545997,  4.0145515
6,  4.0097715 ,
       3.9965272 ,  3.96756132,  3.89743337,  0.        ]), array([-0.17390746,  4.13452045,  4.26900034,  4.0333876
7,  3.98192923,
       4.07079639,  3.94729996,  4.01727144,  0.        ]), array([-0.07055162,  4.04929216,  4.02046342,  4.1009027
6,  4.04287828,
       4.04545593,  3.92604793,  4.12262731,  0.        ]), array([-0.07490431,  3.99114487,  4.08220271,  4.0990081
,  4.09600509,
       4.13488569,  4.12621295,  4.21282454,  0.        ]), array([-0.2382292 ,  4.06077952,  4.1184042 ,  4.0342154
2,  4.1470983 ,
       4.12841635,  4.06101313,  4.08840404,  0.        ]), array([-0.19266037,  4.07793893,  4.04033203,  4.0766296
4,  4.12850526,
       4.05046172,  4.05144254,  4.14084799,  0.        ]), array([0.41694113, 3.94235853, 4.19440514, 4.22225271,
3.89607035,
       3.98028884, 3.99791016, 3.91351085, 0.        ])]], 'Retail': [[array([0.17945357, 3.89016868, 3.97636201, 4.0
099118 , 3.93635193,
       3.84195771, 3.99985443, 3.84400959, 0.        ]), array([-0.3765591 ,  3.8293892 ,  3.9749349 ,  3.89304377,
3.96943349,
       3.75728854,  3.96401308,  4.05468437,  0.        ]), array([-0.31998854,  3.93182563,  3.83186633,  3.8116689
5,  3.98869485,
       3.82748232,  3.95893916,  3.90074071,  0.        ]), array([-0.11531952,  3.93195265,  3.91338955,  3.8948874
8,  3.80522203,
       4.016334  ,  3.67469779,  3.87726772,  0.        ]), array([-0.2370328 ,  3.88782436,  3.96147072,  3.8996800
9,  3.9138127 ,
       3.81819257,  3.81276022,  3.78291992,  0.        ]), array([-0.07775226,  3.84684672,  3.91733724,  3.9880433
7,  3.92387202,
       3.84077768,  3.94931827,  3.85366465,  0.        ]), array([0.07239315, 3.86089205, 4.09465548, 4.02522786,
3.98848867,
       3.84992876, 3.94926694, 3.69833631, 0.        ]), array([-0.50164019,  4.08962228,  3.97399387,  4.10713935,
4.15954334,
       4.06052585,  3.83883972,  3.76471311,  0.        ]), array([-0.23489777,  4.13402767,  4.04249867,  4.1826601
8,  4.14710573,
       3.96474298,  3.88373936,  3.76895805,  0.        ]), array([-0.28431193,  4.30924147,  4.24954044,  4.2109082
,  4.12636343,
       4.01557826,  3.91945444,  3.918824  ,  0.        ]), array([-0.18476449,  4.26998889,  4.24404235,  4.2842503
6,  4.18144776,
       4.06884273,  3.93355076,  3.88935932,  0.        ]), array([-0.15687585,  4.41389469,  4.30838106,  4.4068273
4,  4.24461621,
       4.17473813,  4.0782616 ,  3.92256836,  0.        ]), array([-0.50092712,  4.33599709,  4.37666885,  4.4069446
4,  4.24984188,
       4.27190323,  3.98640803,  3.96636845,  0.        ]), array([-0.2596975 ,  4.47315768,  4.36629972,  4.4033171
4,  4.09721407,
       4.17660297,  4.00369873,  4.02340042,  0.        ]), array([-0.44375793,  4.36090177,  4.50211535,  4.2928392
1,  4.50110913,
       4.31979606,  4.02078041,  4.12823297,  0.        ]), array([-0.29707756,  4.40964242,  4.35493064,  4.3304985
,  4.31869115,
       4.27517188,  4.09137749,  3.98150276,  0.        ]), array([-0.43396147,  4.30985251,  4.32463379,  4.4396652
7,  4.21851377,
       4.18099121,  4.04066643,  4.01184692,  0.        ]), array([-0.21963645,  4.34265823,  4.34803922,  4.2923716
2,  4.17602508,
       4.09995349,  4.0724491 ,  3.95363148,  0.        ]), array([-0.35385978,  4.25708865,  4.25805038,  4.2691066
6,  4.32877247,
       4.17133554,  3.88383999,  4.03319385,  0.        ]), array([-0.32277897,  4.24267067,  4.21769832,  4.1352856
1,  4.21291059,
       3.88221834,  4.07459593,  4.14933824,  0.        ]), array([-0.19198447,  4.21912829,  4.18849211,  4.2513471
9,  4.14750887,
       4.01799359,  4.06942022,  4.00730442,  0.        ]), array([-0.13907755,  3.9960956 ,  4.10541735,  4.0931653
9,  4.05688916,
       3.95879253,  3.92015466,  3.90224734,  0.        ]), array([-0.0460108 ,  4.18620908,  3.99490395,  3.9714339
3,  4.04012148,
       3.88276513,  3.94905701,  3.89343492,  0.        ]), array([-0.48145587,  4.05635939,  3.99409595,  4.0218871
3,  4.04292534,
       3.87576028,  3.81921464,  3.9399648 ,  0.        ])]], 'Parking': [[array([-0.07678979,  4.87039991,  4.86517
019,  4.83449787,  4.84431782,
       4.84425611,  4.86529709,  4.8451585 ,  0.        ]), array([-0.00954917,  4.85006594,  4.83562024,  4.8325946
5,  4.87028817,
       4.87458095,  4.83472584,  4.8563624 ,  0.        ]), array([0.02342221, 4.83532392, 4.82055782, 4.82930086,
4.80314069,
       4.83308891, 4.8445936 , 4.86833911, 0.        ]), array([2.17925196e-03, 4.86103603e+00, 4.85320426e+00, 4.846
46321e+00,
       4.84475355e+00, 4.85301731e+00, 4.86131864e+00, 4.83180589e+00,
       0.00000000e+00]), array([-2.16452348e-03,  4.85759882e+00,  4.85530190e+00,  4.84199502e+00,
        4.81557738e+00,  4.79472080e+00,  4.83971292e+00,  4.86295230e+00,
        0.00000000e+00]), array([0.10561198, 4.79992237, 4.83808955, 4.83576246, 4.86684313,
       4.85625829, 4.8276391 , 4.81984747, 0.        ]), array([-0.03947447,  4.83439364,  4.80906834,  4.83496094,
4.81501756,
       4.81905121,  4.80758902,  4.79086191,  0.        ]), array([-0.08580251,  4.46680632,  4.43006523,  4.4898123
4,  4.48853891,
       4.42911956,  4.42867117,  4.43364003,  0.        ]), array([-0.13876774,  4.23445626,  4.18453452,  4.2106167
,  4.19547694,
       4.19228085,  4.09480794,  4.08748453,  0.        ]), array([-0.15688605,  4.03070469,  4.02403963,  4.0446682
5,  4.06218205,
       4.0862467 ,  3.87027995,  3.87564941,  0.        ]), array([-0.1518098 ,  4.09833979,  4.06710449,  4.0769881
8,  4.06038846,
       4.07532889,  3.9688528 ,  3.92652703,  0.        ]), array([-0.02120609,  3.99577415,  3.96981556,  4.0264054
8,  4.04643762,
       4.03600158,  3.84917959,  3.90314941,  0.        ]), array([-0.07614061,  4.03478194,  3.96636392,  4.0092431
6,  4.02439156,
       3.99587154,  3.87466675,  3.89232021,  0.        ]), array([-0.13921624,  4.02378963,  3.9915444 ,  4.0032617
2,  4.0912858 ,
       4.05519876,  3.86629232,  3.93557632,  0.        ]), array([-0.0353687 ,  3.98442642,  4.07701726,  4.0733211
```

```
,  4.03850955,
        4.02131485,  3.89222838,  3.95887644,  0.        ]), array([-0.07653784,  4.05437599,  4.08544413,  4.0790442
5,  4.11288361,
        4.04671044,  3.89264292,  3.91102783,  0.        ]), array([-0.13657896,  4.14065006,  4.15520101,  4.1286342
1,  4.15155308,
        4.15003835,  4.04299468,  4.01803574,  0.        ]), array([-0.02298457,  4.28968185,  4.28977865,  4.2915016
1,  4.32198517,
        4.27123892,  4.15368413,  4.14946241,  0.        ]), array([0.05687431, 4.40414888, 4.43487669, 4.42213018,
4.4529653 ,
       4.43610717, 4.39555971, 4.36449963, 0.        ]), array([-9.83889122e-04,  4.57583963e+00,  4.59766513e+00,
4.55778724e+00,
        4.59021700e+00,  4.54356505e+00,  4.51789563e+00,  4.56088011e+00,
        0.00000000e+00]), array([3.40690802e-03, 4.79138993e+00, 4.78083737e+00, 4.78017578e+00,
       4.78889630e+00, 4.77349631e+00, 4.71453828e+00, 4.77856445e+00,
       0.00000000e+00]), array([-0.01234177,  4.88424432,  4.89092623,  4.88724772,  4.8743782 ,
        4.84834141,  4.850508  ,  4.85250539,  0.        ]), array([0.01379498, 4.86186736, 4.87752525, 4.90624065,
4.86026443,
       4.88053344, 4.8431761 , 4.83419689, 0.        ]), array([0.02430301, 4.88312026, 4.88027635, 4.86424785, 4.855
12202,
       4.89317676, 4.85938444, 4.85995104, 0.        ])]]]
```

Расчет качества

Используем индивидуальные модели здания, иначе общую модель по всем зданиям данного типа в городе, иначе общую модель по всем зданиям такого типа (по всем городам).

Ввод [17]:
```
def calculate_model (x):
    lr = -1
    model = energy_lr[x.building_id][x.hour]
    if len(model) == 0:
        model = energy_lr_use_site[x.primary_use][x.site_id][x.hour]
    if len(model) == 0:
        model = energy_lr_use[x.primary_use][x.hour]
    if len(model) > 0:
        lr = np.sum([x[col] * model[i] for i,col in enumerate(lr_columns[3:])])
        lr += model[len(lr_columns)-3]
        lr = np.exp(lr)
    if lr < 0:
        lr = 0
    x["meter_reading_lr_q"] = (np.log(x.meter_reading + 1) -
                               np.log(1 + lr))**2
    return x

energy_test = energy_test.apply(calculate_model,
                                axis=1, result_type="expand")
energy_test_lr_rmsle = np.sqrt(energy_test["meter_reading_lr_q"].sum() / len(energy_test))
print ("Качество линейной регрессии, 100 зданий:",
       energy_test_lr_rmsle, round(energy_test_lr_rmsle, 1))
```

```
Качество линейной регрессии, 100 зданий: 0.33775986225141585 0.3
```

ОПТИМИЗАЦИЯ РЕГРЕССИИ

Постановка задачи

Рассмотрим несколько моделей линейной регрессии, чтобы выяснить более оптимальную для первых 20 зданий.

Данные:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz
2. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
import numpy as np
from scipy.interpolate import interp1d
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet, BayesianRidge
```

Загрузка данных, отсечение 20 зданий, объединение и оптимизация

Ввод [2]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        elif col == "timestamp":
            df[col] = pd.to_datetime(df[col])
        elif str(col_type)[:8] != "datetime":
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                            / start_mem, 1), '%)')
    return df
```

Ввод [3]:
```
buildings = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz")
weather = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz")
energy = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz")
energy = energy[(energy["building_id"]<20)]
energy = pd.merge(left=energy, right=buildings, how="left",
                  left_on="building_id", right_on="building_id")
energy = energy.set_index(["timestamp", "site_id"])
weather = weather.set_index(["timestamp", "site_id"])
energy = pd.merge(left=energy, right=weather, how="left",
                  left_index=True, right_index=True)
energy.reset_index(inplace=True)
energy = energy.drop(columns=["meter", "site_id", "year_built",
                              "square_feet", "floor_count"], axis=1)
del buildings
del weather
energy = reduce_mem_usage(energy)
print (energy.info())
```

```
Потребление памяти меньше на 10.39 Мб (минус 70.5 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175680 entries, 0 to 175679
Data columns (total 11 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   timestamp           175680 non-null  datetime64[ns]
 1   building_id         175680 non-null  int8
 2   meter_reading       175680 non-null  float16
 3   primary_use         175680 non-null  category
 4   air_temperature     175620 non-null  float16
 5   cloud_coverage      99080 non-null   float16
 6   dew_temperature     175620 non-null  float16
 7   precip_depth_1_hr   175660 non-null  float16
 8   sea_level_pressure  173980 non-null  float16
 9   wind_direction      170680 non-null  float16
 10  wind_speed          175680 non-null  float16
dtypes: category(1), datetime64[ns](1), float16(8), int8(1)
memory usage: 4.4 MB
None
```

Обогащение данных: час, дни недели, праздники, логарифм

Ввод [4]:
```
energy["hour"] = energy["timestamp"].dt.hour.astype("int8")
energy["weekday"] = energy["timestamp"].dt.weekday.astype("int8")
for weekday in range(0,7):
    energy['is_wday' + str(weekday)] = energy['weekday'].isin([weekday]).astype("int8")
energy["date"] = pd.to_datetime(energy["timestamp"].dt.date)
dates_range = pd.date_range(start='2015-12-31', end='2017-01-01')
us_holidays = calendar().holidays(start=dates_range.min(),
                                  end=dates_range.max())
energy['is_holiday'] = energy['date'].isin(us_holidays).astype("int8")
energy["meter_reading_log"] = np.log(energy["meter_reading"] + 1)
```

Разделение данных

Ввод [5]:
```
energy_train, energy_test = train_test_split(energy[energy["meter_reading"]>0], test_size=0.2)
print (energy_train.head())
```

```
                timestamp  building_id  meter_reading primary_use  \
136959 2016-10-12 07:00:00           19      257.25000      Office
174959 2016-12-30 11:00:00           19      221.12500      Office
87471  2016-07-01 05:00:00           11      484.50000   Education
158734 2016-11-26 16:00:00           14      454.00000   Education
168569 2016-12-17 04:00:00            9       63.46875      Office

        air_temperature  cloud_coverage  dew_temperature  precip_depth_1_hr  \
136959        21.703125             2.0        20.000000                0.0
174959        10.601562             NaN        -3.900391                0.0
87471         25.000000             NaN        23.296875                0.0
158734        23.906250             NaN        15.000000                0.0
168569        18.296875             0.0        15.000000                0.0

        sea_level_pressure  wind_direction  ...  is_wday0  is_wday1  is_wday2  \
136959              1018.0           350.0  ...         0         0         1
174959              1022.5           330.0  ...         0         0         0
87471               1017.0           170.0  ...         0         0         0
158734              1021.0           330.0  ...         0         0         0
168569              1022.5           120.0  ...         0         0         0

        is_wday3  is_wday4  is_wday5  is_wday6       date  is_holiday  \
136959         0         0         0         0 2016-10-12           0
174959         0         1         0         0 2016-12-30           0
87471          0         1         0         0 2016-07-01           0
158734         0         0         1         0 2016-11-26           0
168569         0         0         1         0 2016-12-17           0

        meter_reading_log
136959           5.554688
174959           5.402344
87471            6.183594
158734           6.121094
168569           4.167969

[5 rows x 23 columns]
```

Линейная регрессия: по часам

Ввод [6]:
```
from sklearn.metrics import r2_score
```

Ввод [7]:
```
hours = range(0, 24)
buildings = range(0, energy_train["building_id"].max() + 1)
lr_columns = ["meter_reading_log", "hour", "building_id", "is_holiday"]
for wday in range(0,7):
    lr_columns.append("is_wday" + str(wday))
lr_models = {
    "LinearRegression": LinearRegression,
    "Lasso-0.01": Lasso,
    "Lasso-0.1": Lasso,
    "Lasso-1.0": Lasso,
    "Ridge-0.01": Ridge,
    "Ridge-0.1": Ridge,
    "Ridge-1.0": Ridge,
    "ElasticNet-1-1": ElasticNet,
    "ElasticNet-0.1-1": ElasticNet,
    "ElasticNet-1-0.1": ElasticNet,
    "ElasticNet-0.1-0.1": ElasticNet,
    "BayesianRidge": BayesianRidge
}
energy_train_lr = pd.DataFrame(energy_train, columns=lr_columns)
```

Линейная регрессия

$$z = Ax + By + C, |z - z_0|^2 \to min$$

Лассо + LARS Лассо

$$\frac{1}{2n}|z - z_0|^2 + a(|A| + |B|) \to min$$

Гребневая регрессия

$$|z - z_0|^2 + a(A^2 + B^2) \to min$$

ElasticNet: Лассо + Гребневая регрессия

$$\frac{1}{2n}|z - z_0|^2 + \alpha p|A^2 + B^2| + (\alpha - p)(|A| + |B|)/2 \to min$$

Ввод [8]:
```
lr_models_scores = {}
for _ in lr_models:
    lr_model = lr_models[_]
    energy_lr_scores = [[]]*len(buildings)
    for building in buildings:
        energy_lr_scores[building] = [0]*len(hours)
        energy_train_b = energy_train_lr[energy_train_lr["building_id"]==building]
        for hour in hours:
            energy_train_bh = energy_train_b[energy_train_b["hour"]==hour]
            y = energy_train_bh["meter_reading_log"]
            x = energy_train_bh.drop(labels=["meter_reading_log",
                "hour", "building_id"], axis=1)
            if _ in ["Ridge-0.1", "Lasso-0.1"]:
                model = lr_model(alpha=.1, fit_intercept=False).fit(x, y)
            elif _ in ["Ridge-0.01", "Lasso-0.01"]:
                model = lr_model(alpha=.01, fit_intercept=False).fit(x, y)
            elif _ == "ElasticNet-1-1":
                model = lr_model(alpha=1, l1_ratio=1, fit_intercept=False).fit(x, y)
            elif _ == "ElasticNet-1-0.1":
                model = lr_model(alpha=1, l1_ratio=.1, fit_intercept=False).fit(x, y)
            elif _ == "ElasticNet-0.1-1":
                model = lr_model(alpha=.1, l1_ratio=1, fit_intercept=False).fit(x, y)
            elif _ == "ElasticNet-0.1-0.1":
                model = lr_model(alpha=.1, l1_ratio=.05, fit_intercept=False).fit(x, y)
            else:
                model = lr_model(fit_intercept=False).fit(x, y)
            energy_lr_scores[building][hour] = r2_score(y, model.predict(x))
    lr_models_scores[_] = np.mean(energy_lr_scores)
```

Ввод [9]:
```
print (lr_models_scores)
```

```
{'LinearRegression': 0.1336053539217353, 'Lasso-0.01': -0.19569305108000243, 'Lasso-0.1': -31.764987418282185, 'Lasso
-1.0': -2483.013230734368, 'Ridge-0.01': 0.13317572160319768, 'Ridge-0.1': 0.09101491880338457, 'Ridge-1.0': -3.78970
33516477063, 'ElasticNet-1-1': -2483.013230734368, 'ElasticNet-0.1-1': -31.764987418282185, 'ElasticNet-1-0.1': -206
6.2678757946846, 'ElasticNet-0.1-0.1': -433.4675202627676, 'BayesianRidge': 0.13359551755117657}
```

Проверим модели: LinearRegression, Lasso, BayesianRidge

Ввод [10]:
```
energy_lr = [[]]*len(buildings)
energy_lasso = [[]]*len(buildings)
energy_br = [[]]*len(buildings)
for building in buildings:
    energy_train_b = energy_train_lr[energy_train_lr["building_id"]==building]
    energy_lr[building] = [[]]*len(hours)
    energy_lasso[building] = [[]]*len(hours)
    energy_br[building] = [[]]*len(hours)
    for hour in hours:
        energy_train_bh = pd.DataFrame(energy_train_b[energy_train_b["hour"]==hour])
        y = energy_train_bh["meter_reading_log"]
        if len(y) > 0:
            x = energy_train_bh.drop(labels=["meter_reading_log",
                "hour", "building_id"], axis=1)
            model = LinearRegression(fit_intercept=False).fit(x, y)
            energy_lr[building][hour] = np.append([model.coef_], model.intercept_)
            model = Lasso(fit_intercept=False, alpha=.01).fit(x, y)
            energy_lasso[building][hour] = np.append([model.coef_], model.intercept_)
            model = BayesianRidge(fit_intercept=False).fit(x, y)
            energy_br[building][hour] = np.append([model.coef_], model.intercept_)
print (energy_lr[0][0])
print (energy_lasso[0][0])
print (energy_br[0][0])
```

```
[-0.0515393   5.43848036  5.46859976  5.47509766  5.46080633  5.51341712
  5.43010603  5.4         0.        ]
[0.         5.3671412  5.40244591 5.40343099 5.38699219 5.43863451
 5.36867746 5.314      0.        ]
[-0.05100353  5.43796645  5.46814601  5.47460551  5.46029315  5.51289999
  5.42968765  5.39941754  0.        ]
```

ЭКСПОРТ И ИМПОРТ ДАННЫХ

Постановка задачи

Подготовим данные для построения модели: получим, объединим, оптимизируем и обогатим данные.

Сохраним готовые данные в нескольких форматах: CSV, HDF5

Данные:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz
2. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
import numpy as np
from scipy.interpolate import interp1d
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import os
```

Загрузка данных, отсечение 20 зданий, объединение и оптимизация

Ввод [2]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        elif col == "timestamp":
            df[col] = pd.to_datetime(df[col])
        elif str(col_type)[:8] != "datetime":
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                            / start_mem, 1), '%)')
    return df
```

Ввод [3]:
```
buildings = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz")
weather = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz")
weather = weather[weather["site_id"] == 0]
energy = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz")
energy = energy[energy["building_id"] < 20]
energy = pd.merge(left=energy, right=buildings, how="left",
                  left_on="building_id", right_on="building_id")
del buildings
```

```
Потребление памяти меньше на 10.39 Мб (минус 70.5 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175680 entries, 0 to 175679
Data columns (total 11 columns):
```

```
timestamp             175680 non-null datetime64[ns]
building_id           175680 non-null int8
meter_reading         175680 non-null float16
primary_use           175680 non-null category
air_temperature       175620 non-null float16
cloud_coverage        99080 non-null float16
dew_temperature       175620 non-null float16
precip_depth_1_hr     175660 non-null float16
sea_level_pressure    173980 non-null float16
wind_direction        170680 non-null float16
wind_speed            175680 non-null float16
dtypes: category(1), datetime64[ns](1), float16(8), int8(1)
memory usage: 4.4 MB
None
```

Интерполяция значений

Ввод [4]:
```
weather["precip_depth_1_hr"] = weather["precip_depth_1_hr"].apply(lambda x:x if x>0 else 0)
interpolate_columns = ["air_temperature", "dew_temperature",
                       "cloud_coverage", "wind_speed", "wind_direction",
                       "precip_depth_1_hr", "sea_level_pressure"]
for col in interpolate_columns:
    weather[col] = weather[col].interpolate(limit_direction='both',
                          kind='cubic')
```

Обогащение данных: погода

Ввод [ ]:
```
weather["air_temperature_diff1"] = weather["air_temperature"].diff()
weather.at[0, "air_temperature_diff1"] = weather.at[1, "air_temperature_diff1"]
weather["air_temperature_diff2"] = weather["air_temperature_diff1"].diff()
weather.at[0, "air_temperature_diff2"] = weather.at[1, "air_temperature_diff2"]
```

Объединение погодных данных

Ввод [ ]:
```
energy = energy.set_index(["timestamp", "site_id"])
weather = weather.set_index(["timestamp", "site_id"])
energy = pd.merge(left=energy, right=weather, how="left",
                  left_index=True, right_index=True)
energy.reset_index(inplace=True)
energy = energy.drop(columns=["meter", "site_id", "year_built",
                              "square_feet", "floor_count"], axis=1)
energy = reduce_mem_usage(energy)
del weather
print (energy.info())
```

Обогащение данных: дата

Ввод [5]:
```
energy["hour"] = energy["timestamp"].dt.hour.astype("int8")
energy["weekday"] = energy["timestamp"].dt.weekday.astype("int8")
energy["week"] = energy["timestamp"].dt.week.astype("int8")
energy["month"] = energy["timestamp"].dt.month.astype("int8")
energy["date"] = pd.to_datetime(energy["timestamp"].dt.date)
dates_range = pd.date_range(start='2015-12-31', end='2017-01-01')
us_holidays = calendar().holidays(start=dates_range.min(),
                                  end=dates_range.max())
energy['is_holiday'] = energy['date'].isin(us_holidays).astype("int8")
for weekday in range(0,7):
    energy['is_wday' + str(weekday)] = energy['weekday'].isin([weekday]).astype("int8")
for week in range(1,54):
    energy['is_w' + str(week)] = energy['week'].isin([week]).astype("int8")
for month in range(1,13):
    energy['is_m' + str(month)] = energy['month'].isin([month]).astype("int8")
```

Логарифмирование данных

z = A * x + B * y -> log z = A * x + B * y => z = e^Ax * e^By => z = a^x * b^y

```
Ввод [7]: energy["meter_reading_log"] = np.log(energy["meter_reading"] + 1)
```


```
Ввод [7]: energy["meter_reading_log"] = np.log(energy["meter_reading"] + 1)
```


Экспорт данных в CSV и HDF5

```
Ввод [8]: print (energy.info())
          energy.to_csv("energy.0-20.ready.csv.gz", index=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175680 entries, 0 to 175679
Data columns (total 92 columns):
timestamp                175680 non-null datetime64[ns]
building_id              175680 non-null int8
meter_reading            175680 non-null float16
primary_use              175680 non-null category
air_temperature          175680 non-null float16
cloud_coverage           175680 non-null float16
dew_temperature          175680 non-null float16
precip_depth_1_hr        175680 non-null float64
sea_level_pressure       175680 non-null float16
wind_direction           175680 non-null float16
wind_speed               175680 non-null float16
hour                     175680 non-null int8
weekday                  175680 non-null int8
week                     175680 non-null int8
month                    175680 non-null int8
date                     175680 non-null datetime64[ns]
is_holiday               175680 non-null int8
is_wday0                 175680 non-null int8
is_wday1                 175680 non-null int8
is_wday2                 175680 non-null int8
is_wday3                 175680 non-null int8
is_wday4                 175680 non-null int8
is_wday5                 175680 non-null int8
is_wday6                 175680 non-null int8
is_w1                    175680 non-null int8
is_w2                    175680 non-null int8
is_w3                    175680 non-null int8
is_w4                    175680 non-null int8
is_w5                    175680 non-null int8
is_w6                    175680 non-null int8
is_w7                    175680 non-null int8
is_w8                    175680 non-null int8
is_w9                    175680 non-null int8
is_w10                   175680 non-null int8
is_w11                   175680 non-null int8
is_w12                   175680 non-null int8
is_w13                   175680 non-null int8
```

```
is_w14                  175680 non-null int8
is_w15                  175680 non-null int8
is_w16                  175680 non-null int8
is_w17                  175680 non-null int8
is_w18                  175680 non-null int8
is_w19                  175680 non-null int8
is_w20                  175680 non-null int8
is_w21                  175680 non-null int8
is_w22                  175680 non-null int8
is_w23                  175680 non-null int8
is_w24                  175680 non-null int8
is_w25                  175680 non-null int8
is_w26                  175680 non-null int8
is_w27                  175680 non-null int8
is_w28                  175680 non-null int8
is_w29                  175680 non-null int8
is_w30                  175680 non-null int8
is_w31                  175680 non-null int8
is_w32                  175680 non-null int8
is_w33                  175680 non-null int8
is_w34                  175680 non-null int8
is_w35                  175680 non-null int8
is_w36                  175680 non-null int8
is_w37                  175680 non-null int8
is_w38                  175680 non-null int8
is_w39                  175680 non-null int8
is_w40                  175680 non-null int8
is_w41                  175680 non-null int8
is_w42                  175680 non-null int8
is_w43                  175680 non-null int8
is_w44                  175680 non-null int8
is_w45                  175680 non-null int8
is_w46                  175680 non-null int8
is_w47                  175680 non-null int8
is_w48                  175680 non-null int8
is_w49                  175680 non-null int8
is_w50                  175680 non-null int8
is_w51                  175680 non-null int8
is_w52                  175680 non-null int8
is_w53                  175680 non-null int8
is_m1                   175680 non-null int8
is_m2                   175680 non-null int8
is_m3                   175680 non-null int8
is_m4                   175680 non-null int8
is_m5                   175680 non-null int8
is_m6                   175680 non-null int8
is_m7                   175680 non-null int8
is_m8                   175680 non-null int8
is_m9                   175680 non-null int8
is_m10                  175680 non-null int8
is_m11                  175680 non-null int8
is_m12                  175680 non-null int8
air_temperature_diff1   175680 non-null float16
air_temperature_diff2   175680 non-null float16
meter_reading_log       175680 non-null float16
dtypes: category(1), datetime64[ns](2), float16(10), float64(1), int8(78)
memory usage: 20.6 MB
None
```

Ввод [9]:
```
energy = pd.read_csv("energy.0-20.ready.csv.gz")
print (energy.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175680 entries, 0 to 175679
Data columns (total 92 columns):
timestamp                175680 non-null object
building_id              175680 non-null int64
meter_reading            175680 non-null float64
primary_use              175680 non-null object
air_temperature          175680 non-null float64
cloud_coverage           175680 non-null float64
dew_temperature          175680 non-null float64
precip_depth_1_hr        175680 non-null float64
sea_level_pressure       175680 non-null float64
wind_direction           175680 non-null float64
wind_speed               175680 non-null float64
hour                     175680 non-null int64
weekday                  175680 non-null int64
week                     175680 non-null int64
month                    175680 non-null int64
date                     175680 non-null object
is_holiday               175680 non-null int64
is_wday0                 175680 non-null int64
is_wday1                 175680 non-null int64
is_wday2                 175680 non-null int64
is_wday3                 175680 non-null int64
is_wday4                 175680 non-null int64
is_wday5                 175680 non-null int64
is_wday6                 175680 non-null int64
is_w1                    175680 non-null int64
is_w2                    175680 non-null int64
is_w3                    175680 non-null int64
is_w4                    175680 non-null int64
is_w5                    175680 non-null int64
is_w6                    175680 non-null int64
is_w7                    175680 non-null int64
is_w8                    175680 non-null int64
is_w9                    175680 non-null int64
is_w10                   175680 non-null int64
is_w11                   175680 non-null int64
is_w12                   175680 non-null int64
is_w13                   175680 non-null int64
is_w14                   175680 non-null int64
is_w15                   175680 non-null int64
is_w16                   175680 non-null int64
is_w17                   175680 non-null int64
is_w18                   175680 non-null int64
is_w19                   175680 non-null int64
is_w20                   175680 non-null int64
is_w21                   175680 non-null int64
is_w22                   175680 non-null int64
is_w23                   175680 non-null int64
is_w24                   175680 non-null int64
is_w25                   175680 non-null int64
is_w26                   175680 non-null int64
```

```
is_w27                    175680 non-null int64
is_w28                    175680 non-null int64
is_w29                    175680 non-null int64
is_w30                    175680 non-null int64
is_w31                    175680 non-null int64
is_w32                    175680 non-null int64
is_w33                    175680 non-null int64
is_w34                    175680 non-null int64
is_w35                    175680 non-null int64
is_w36                    175680 non-null int64
is_w37                    175680 non-null int64
is_w38                    175680 non-null int64
is_w39                    175680 non-null int64
is_w40                    175680 non-null int64
is_w41                    175680 non-null int64
is_w42                    175680 non-null int64
is_w43                    175680 non-null int64
is_w44                    175680 non-null int64
is_w45                    175680 non-null int64
is_w46                    175680 non-null int64
is_w47                    175680 non-null int64
is_w48                    175680 non-null int64
is_w49                    175680 non-null int64
is_w50                    175680 non-null int64
is_w51                    175680 non-null int64
is_w52                    175680 non-null int64
is_w53                    175680 non-null int64
is_m1                     175680 non-null int64
is_m2                     175680 non-null int64
is_m3                     175680 non-null int64
is_m4                     175680 non-null int64
is_m5                     175680 non-null int64
is_m6                     175680 non-null int64
is_m7                     175680 non-null int64
is_m8                     175680 non-null int64
is_m9                     175680 non-null int64
is_m10                    175680 non-null int64
is_m11                    175680 non-null int64
is_m12                    175680 non-null int64
air_temperature_diff1     175680 non-null float64
air_temperature_diff2     175680 non-null float64
meter_reading_log         175680 non-null float64
dtypes: float64(11), int64(78), object(3)
memory usage: 123.3+ MB
None
```

Экспорт данных в HDF5

HDF5: / ->

- Группа (+ метаданные)
  - Набор данных
  - ...

Ввод [20]:
```
energy = reduce_mem_usage(energy)
energy.to_hdf('energy.0-20.ready.h5', "energy", format='table',
              compression="gzip", complevel=9, mode="w")
print ("CSV:", os.path.getsize(os.getcwd() + '\energy.0-20.ready.csv.gz'))
print ("HDF5:", os.path.getsize(os.getcwd() + '\energy.0-20.ready.h5'))
```

```
Потребление памяти меньше на 0.0 Мб (минус 0.0 %)
CSV: 1749050
HDF5: 914546
```

Ввод [11]:
```
energy = pd.read_hdf('energy.0-20.ready.h5', "energy")
print (energy.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 175680 entries, 0 to 175679
Data columns (total 92 columns):
timestamp                175680 non-null datetime64[ns]
building_id              175680 non-null int8
meter_reading            175680 non-null float16
primary_use              175680 non-null category
air_temperature          175680 non-null float16
cloud_coverage           175680 non-null float16
dew_temperature          175680 non-null float16
precip_depth_1_hr        175680 non-null float16
sea_level_pressure       175680 non-null float16
wind_direction           175680 non-null float16
wind_speed               175680 non-null float16
hour                     175680 non-null int8
weekday                  175680 non-null int8
week                     175680 non-null int8
month                    175680 non-null int8
date                     175680 non-null category
is_holiday               175680 non-null int8
is_wday0                 175680 non-null int8
is_wday1                 175680 non-null int8
is_wday2                 175680 non-null int8
is_wday3                 175680 non-null int8
is_wday4                 175680 non-null int8
is_wday5                 175680 non-null int8
is_wday6                 175680 non-null int8
is_w1                    175680 non-null int8
is_w2                    175680 non-null int8
is_w3                    175680 non-null int8
is_w4                    175680 non-null int8
is_w5                    175680 non-null int8
is_w6                    175680 non-null int8
is_w7                    175680 non-null int8
is_w8                    175680 non-null int8
is_w9                    175680 non-null int8
is_w10                   175680 non-null int8
is_w11                   175680 non-null int8
is_w12                   175680 non-null int8
is_w13                   175680 non-null int8
is_w14                   175680 non-null int8
is_w15                   175680 non-null int8
is_w16                   175680 non-null int8
is_w17                   175680 non-null int8
is_w18                   175680 non-null int8
is_w19                   175680 non-null int8
is_w20                   175680 non-null int8
is_w21                   175680 non-null int8
is_w22                   175680 non-null int8
is_w23                   175680 non-null int8
is_w24                   175680 non-null int8
is_w25                   175680 non-null int8
is_w26                   175680 non-null int8
```

```
is_w27                    175680 non-null int8
is_w28                    175680 non-null int8
is_w29                    175680 non-null int8
is_w30                    175680 non-null int8
is_w31                    175680 non-null int8
is_w32                    175680 non-null int8
is_w33                    175680 non-null int8
is_w34                    175680 non-null int8
is_w35                    175680 non-null int8
is_w36                    175680 non-null int8
is_w37                    175680 non-null int8
is_w38                    175680 non-null int8
is_w39                    175680 non-null int8
is_w40                    175680 non-null int8
is_w41                    175680 non-null int8
is_w42                    175680 non-null int8
is_w43                    175680 non-null int8
is_w44                    175680 non-null int8
is_w45                    175680 non-null int8
is_w46                    175680 non-null int8
is_w47                    175680 non-null int8
is_w48                    175680 non-null int8
is_w49                    175680 non-null int8
is_w50                    175680 non-null int8
is_w51                    175680 non-null int8
is_w52                    175680 non-null int8
is_w53                    175680 non-null int8
is_m1                     175680 non-null int8
is_m2                     175680 non-null int8
is_m3                     175680 non-null int8
is_m4                     175680 non-null int8
is_m5                     175680 non-null int8
is_m6                     175680 non-null int8
is_m7                     175680 non-null int8
is_m8                     175680 non-null int8
is_m9                     175680 non-null int8
is_m10                    175680 non-null int8
is_m11                    175680 non-null int8
is_m12                    175680 non-null int8
air_temperature_diff1     175680 non-null float16
air_temperature_diff2     175680 non-null float16
meter_reading_log         175680 non-null float16
dtypes: category(2), datetime64[ns](1), float16(11), int8(78)
memory usage: 20.0 MB
None
```

Разделение данных и экспорт в HDF5

Ввод [12]:
```
energy_train, energy_test = train_test_split(energy[energy["meter_reading"]>0], test_size=0.2)
print (energy_train.head())
```

```
                  timestamp  building_id  meter_reading  \
114227 2016-08-25 23:00:00            7     474.750000
137382 2016-10-13 05:00:00            2      22.109375
143803 2016-10-26 14:00:00            3     457.750000
121230 2016-09-09 13:00:00           10    2054.000000
106314 2016-08-09 11:00:00           14     426.000000
```

```
                        primary_use  air_temperature  cloud_coverage  \
114227                    Education        30.000000        4.000000
137382                    Education        22.796875        6.230469
143803                    Education        25.000000        2.000000
121230  Entertainment/public assembly        27.796875        4.000000
106314                    Education        24.406250        7.605469

        dew_temperature  precip_depth_1_hr  sea_level_pressure  \
114227        21.093750                0.0              1016.0
137382        20.593750                0.0              1017.5
143803        17.796875                0.0              1023.5
121230        24.406250                0.0              1017.5
106314        22.796875                0.0              1016.0

        wind_direction  ...  is_m6  is_m7  is_m8  is_m9  is_m10 is_m11  \
114227            50.0  ...      0      0      1      0       0      0
137382            10.0  ...      0      0      0      0       1      0
143803            70.0  ...      0      0      0      0       1      0
121230            60.0  ...      0      0      0      1       0      0
106314             0.0  ...      0      0      1      0       0      0

        is_m12  air_temperature_diff1  air_temperature_diff2  \
114227       0                    0.0                    0.0
137382       0                    0.0                    0.0
143803       0                    0.0                    0.0
121230       0                    0.0                    0.0
106314       0                    0.0                    0.0

        meter_reading_log
114227           6.164062
137382           3.140625
143803           6.128906
121230           7.628906
106314           6.058594

[5 rows x 92 columns]
```

Ввод [19]:
```
pd.set_option('io.hdf.default_format','table')
store = pd.HDFStore('energy.0-20.ready.split.h5',
                    complevel=9, complib='zlib', mode="w")
store["energy_train"] = energy_train
store["energy_test"] = energy_test
store.put("metadata",
          pd.Series(["Набор обогащенных тестовых данных по 20 зданиям"]))
store.close()
print ("HDF5:", os.path.getsize(os.getcwd() + '\energy.0-20.ready.split.h5'))
```

```
HDF5: 3686169
```

Для хранения атрибутов наборов данных также можно использовать

store.get_storer('energy_train').attrs.my_attr

Чтение из HDF5

Ввод [22]:
```
store = pd.HDFStore('energy.0-20.ready.split.h5')
energy_test = store.get("energy_test")[:]
energy_train = store.get("energy_train")[:]
metadata = store.get("metadata")[:]
store.close()
print (metadata[0])
print (energy_train.head())
```

```
Набор обогащенных тестовых данных по 20 зданиям
                timestamp  building_id  meter_reading  \
114227 2016-08-25 23:00:00            7      474.750000
137382 2016-10-13 05:00:00            2       22.109375
```

```
143803 2016-10-26 14:00:00            3     457.750000
121230 2016-09-09 13:00:00           10    2054.000000
106314 2016-08-09 11:00:00           14     426.000000

                         primary_use  air_temperature  cloud_coverage  \
114227                     Education        30.000000        4.000000
137382                     Education        22.796875        6.230469
143803                     Education        25.000000        2.000000
121230  Entertainment/public assembly        27.796875        4.000000
106314                     Education        24.406250        7.605469

        dew_temperature  precip_depth_1_hr  sea_level_pressure  \
114227         21.093750                0.0              1016.0
137382         20.593750                0.0              1017.5
143803         17.796875                0.0              1023.5
121230         24.406250                0.0              1017.5
106314         22.796875                0.0              1016.0

        wind_direction  ...  is_m6  is_m7  is_m8  is_m9  is_m10 is_m11  \
114227            50.0  ...      0      0      1      0       0      0
137382            10.0  ...      0      0      0      0       1      0
143803            70.0  ...      0      0      0      0       1      0
121230            60.0  ...      0      0      0      1       0      0
106314             0.0  ...      0      0      1      0       0      0

        is_m12  air_temperature_diff1  air_temperature_diff2  \
114227       0                    0.0                    0.0
137382       0                    0.0                    0.0
143803       0                    0.0                    0.0
121230       0                    0.0                    0.0
106314       0                    0.0                    0.0

        meter_reading_log
114227           6.164062
137382           3.140625
143803           6.128906
121230           7.628906
106314           6.058594

[5 rows x 92 columns]
```

АНСАМБЛЬ РЕГРЕССИОННЫХ МОДЕЛЕЙ

Постановка задачи

Загрузим подготовленные данные по энергопотреблению первых 20 зданий (building_id от 0 до 19).

Соберем два набора моделей: по дате (праздники, дни недели и т.д.) и по погоде.

Проведем 10 разбиений данных на обучающие/проверочные и выявим оптимальные веса моделей для каждого часа для каждого здания.

Вычислим оптимизированную метрику качества для ансамбля моделей.

Данные:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz

2. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

Загрузка данных 20 зданий из HDF5

Ввод [2]:
```
energy = pd.read_hdf('energy.0-20.ready.h5', "energy")
print (energy.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 175680 entries, 0 to 175679
Data columns (total 92 columns):
timestamp                175680 non-null datetime64[ns]
building_id              175680 non-null int8
meter_reading            175680 non-null float16
primary_use              175680 non-null category
air_temperature          175680 non-null float16
cloud_coverage           175680 non-null float16
dew_temperature          175680 non-null float16
precip_depth_1_hr        175680 non-null float16
sea_level_pressure       175680 non-null float16
wind_direction           175680 non-null float16
wind_speed               175680 non-null float16
hour                     175680 non-null int8
weekday                  175680 non-null int8
week                     175680 non-null int8
month                    175680 non-null int8
date                     175680 non-null category
is_holiday               175680 non-null int8
is_wday0                 175680 non-null int8
is_wday1                 175680 non-null int8
is_wday2                 175680 non-null int8
is_wday3                 175680 non-null int8
is_wday4                 175680 non-null int8
is_wday5                 175680 non-null int8
is_wday6                 175680 non-null int8
is_w1                    175680 non-null int8
is_w2                    175680 non-null int8
is_w3                    175680 non-null int8
is_w4                    175680 non-null int8
is_w5                    175680 non-null int8
is_w6                    175680 non-null int8
is_w7                    175680 non-null int8
is_w8                    175680 non-null int8
is_w9                    175680 non-null int8
is_w10                   175680 non-null int8
```

```
is_w11                   175680 non-null int8
is_w12                   175680 non-null int8
is_w13                   175680 non-null int8
is_w14                   175680 non-null int8
is_w15                   175680 non-null int8
is_w16                   175680 non-null int8
is_w17                   175680 non-null int8
is_w18                   175680 non-null int8
is_w19                   175680 non-null int8
is_w20                   175680 non-null int8
is_w21                   175680 non-null int8
is_w22                   175680 non-null int8
is_w23                   175680 non-null int8
is_w24                   175680 non-null int8
is_w25                   175680 non-null int8
is_w26                   175680 non-null int8
is_w27                   175680 non-null int8
is_w28                   175680 non-null int8
is_w29                   175680 non-null int8
is_w30                   175680 non-null int8
is_w31                   175680 non-null int8
is_w32                   175680 non-null int8
is_w33                   175680 non-null int8
is_w34                   175680 non-null int8
is_w35                   175680 non-null int8
is_w36                   175680 non-null int8
is_w37                   175680 non-null int8
is_w38                   175680 non-null int8
is_w39                   175680 non-null int8
is_w40                   175680 non-null int8
is_w41                   175680 non-null int8
is_w42                   175680 non-null int8
is_w43                   175680 non-null int8
is_w44                   175680 non-null int8
is_w45                   175680 non-null int8
is_w46                   175680 non-null int8
is_w47                   175680 non-null int8
is_w48                   175680 non-null int8
is_w49                   175680 non-null int8
is_w50                   175680 non-null int8
is_w51                   175680 non-null int8
is_w52                   175680 non-null int8
is_w53                   175680 non-null int8
is_m1                    175680 non-null int8
is_m2                    175680 non-null int8
is_m3                    175680 non-null int8
is_m4                    175680 non-null int8
is_m5                    175680 non-null int8
is_m6                    175680 non-null int8
is_m7                    175680 non-null int8
is_m8                    175680 non-null int8
is_m9                    175680 non-null int8
is_m10                   175680 non-null int8
is_m11                   175680 non-null int8
is_m12                   175680 non-null int8
air_temperature_diff1    175680 non-null float16
```

```
air_temperature_diff2    175680 non-null float16
meter_reading_log        175680 non-null float16
dtypes: category(2), datetime64[ns](1), float16(11), int8(78)
memory usage: 20.0 MB
None
```

Обозначим набор параметров для каждой модели

Ввод [3]:
```
lr_weather_columns = ["meter_reading_log", "hour", "building_id",
                      "air_temperature", "dew_temperature",
                      "sea_level_pressure", "wind_speed",
                      "air_temperature_diff1", "air_temperature_diff2",
                      "cloud_coverage"]
lr_days_columns = ["meter_reading_log", "hour", "building_id", "is_holiday"]
for wday in range(0,7):
    lr_days_columns.append("is_wday" + str(wday))
for week in range(1,54):
    lr_days_columns.append("is_w" + str(week))
for month in range(1,13):
    lr_days_columns.append("is_m" + str(month))
hours = range(0, 24)
buildings = range(0, energy["building_id"].max() + 1)
```

Введем функцию для вычисления качества моделей

Ввод [4]:
```
def calculate_model (x, df_lr, lr_columns):
    lr = -1
    model = df_lr[x.building_id][x.hour]
    if len(model) > 0:
        lr = np.sum([x[col] * model[i] for i,col in enumerate(lr_columns[3:])])
        lr += model[len(lr_columns)-3]
        lr = np.exp(lr)
    if lr < 0 or lr*lr == lr:
        lr = 0
    x["meter_reading_lr_q"] = (np.log(x.meter_reading + 1) -
                               np.log(1 + lr))**2
    return x
```

Введем функции для разделения данных, построение моделей и вычисления их качества (для обновления весов ансамбля)

Ансамбль моделей линейной регрессии: Z = A * погода + B * дни_недели, A + B = 1.

Ввод [5]:
```
def train_model (df, columns):
    df_train_lr = pd.DataFrame(df, columns=columns)
    df_lr = [[]]*len(buildings)
    for building in buildings:
        df_lr[building] = [[]]*len(hours)
        df_train_b = df_train_lr[df_train_lr["building_id"]==building]
        for hour in hours:
            df_train_bh = df_train_b[df_train_b["hour"]==hour]
            y = df_train_bh["meter_reading_log"]
            x = df_train_bh.drop(labels=["meter_reading_log",
                "hour", "building_id"], axis=1)
            model = LinearRegression(fit_intercept=False).fit(x, y)
            df_lr[building][hour] = model.coef_
            df_lr[building][hour] = np.append(df_lr[building][hour], model.intercept_)
    return df_lr

def calculate_weights_model(df_test, df_train, lr_columns):
    df_test = df_test.apply(calculate_model,
```

```
        axis=1, result_type="expand",
        df_lr=train_model(df_train, lr_columns),
        lr_columns=lr_columns)
    return pd.Series(df_test.groupby(["hour",
                        "building_id"]).sum()["meter_reading_lr_q"])

def calculate_weights():
    df_train, df_test = train_test_split(energy[energy["meter_reading"]>0], test_size=0.2)
    return (calculate_weights_model(df_test, df_train, lr_weather_columns),
            calculate_weights_model(df_test, df_train, lr_days_columns))
```

Рассчитаем оптимальные веса для каждого часа и здания

10 раз разобьем исходный набор данных на обучающую/тестовую выборку, рассчитаем в каждом случае значения ошибки для каждого здания и часа.

Сформируем список весов: 1 – учитываем регрессию по дням недели, 0 – учитываем регрессию по погоде

Ввод [6]:
```
weights_weather = []
weights_days = []
for i in range(0, 10):
    print ("Расчет весов ансамбля, итерация", i)
    weights_weather_model, weights_days_model = calculate_weights()
    if len(weights_weather) > 0:
        weights_weather = weights_weather + weights_weather_model
    else:
        weights_weather = weights_weather_model
    if len(weights_days) > 0:
        weights_days = weights_days + weights_days_model
    else:
        weights_days = weights_days_model
weights = [0]*len(buildings)
for b in buildings:
    weights[b] = [0]*len(hours)
    for h in hours:
        if weights_weather.loc[h].at[b] > weights_days.loc[h].at[b]:
            weights[b][h] = 1
print (weights)
```

```
Расчет весов ансамбля, итерация 0
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow
encountered in exp
  import sys
Расчет весов ансамбля, итерация 1
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow
encountered in exp
  import sys
Расчет весов ансамбля, итерация 2
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow
encountered in exp
  import sys
Расчет весов ансамбля, итерация 3
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow
encountered in exp
  import sys
Расчет весов ансамбля, итерация 4
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow
encountered in exp
  import sys
Расчет весов ансамбля, итерация 5
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow
encountered in exp
  import sys
```

```
Расчет весов ансамбля, итерация 6

c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow
encountered in exp
  import sys

Расчет весов ансамбля, итерация 7

c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow
encountered in exp
  import sys

Расчет весов ансамбля, итерация 8

c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow
encountered in exp
  import sys

Расчет весов ансамбля, итерация 9

c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow
encountered in exp
  import sys

[[1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1], [0, 1, 0, 1, 1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1], [1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1],
[0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0], [0, 1, 1, 1, 1, 0, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
```

Посчитаем ансамбль линейной регрессии

Разделим данные на обучающие/тестовые

Ввод [7]:
```
energy_train, energy_test = train_test_split(energy[energy["meter_reading"]>0], test_size=0.2)
```

Обучим модели линейной регрессии по дате/погоде

Ввод [8]:
```
energy_lr_days = train_model(energy_train, lr_days_columns)
energy_lr_weather = train_model(energy_train, lr_weather_columns)
```

Рассчитаем финальное качество ансамбля

Если вес 1, то считаем регрессию по дням недели, если 0 – то по погоде

Ввод [11]:
```
def calculate_model_ensemble (x, model, columns):
    lr = -1
    if len(model) > 0:
        lr = np.sum([x[col] * model[i] for i,col in enumerate(columns[3:])])
        lr += model[len(columns)-3]
        lr = np.exp(lr)
    if lr < 0 or lr*lr == lr:
        lr = 0
    return lr

def calculate_models_ensemble (x):
    lr_d = calculate_model_ensemble(x,
           energy_lr_days[x.building_id][x.hour],
           lr_days_columns)
    lr_w = calculate_model_ensemble(x,
           energy_lr_weather[x.building_id][x.hour],
           lr_weather_columns)
    if weights[x.building_id][x.hour] == 1:
        lr = lr_d
    else:
        lr = lr_w
    lr_sum = (lr_w + lr_d)/2
    x["meter_reading_lr_q"] = (np.log(x.meter_reading + 1) -
                               np.log(1 + lr))**2
    x["meter_reading_sum_q"] = (np.log(x.meter_reading + 1) -
                               np.log(1 + lr_sum))**2
    return x
```

В теории, в идеальном случае, ансамбль линейной регрессии не должен давать никакого преимущества, потому что, в случае, когда:

$$z_1 = Ax = By + C,$$

$$z_2 = Ds + Et + F, \text{ то:}$$

$$z = \alpha z_1 + \beta z_2 = \alpha Ax + \beta By + \alpha C + \beta Ds + \beta Et + \beta F =$$

$$= A_1 x + B_1 y + D_1 s + E_1 t + F_1.$$

И, по сути, ансамбль линейной регрессии – это просто линейная регрессия по всем переменным. Но при использовании небольших наборов (чтобы исключить переобучение) связанных переменных для разных моделей регрессии можно получить небольшой выигрыш.

Ансамбль регрессии в нашем случае не дает никакого улучшения относительно регрессии по совокупному набору параметров.

Однако, использование усредненной суммы показателей каждой конкретной модели дало выигрыш порядка 6 % относительно модели по всем показателям. В этом случае сумму моделей линейной регрессии «компенсирует» ошибки каждой конкретной модели и работает точнее.

Ввод [12]:
```
energy_test = energy_test.apply(calculate_models_ensemble,
                                axis=1, result_type="expand")
energy_test_lr_rmsle = np.sqrt(energy_test["meter_reading_lr_q"].sum() / len(energy_test))
energy_test_sum_rmsle = np.sqrt(energy_test["meter_reading_sum_q"].sum() / len(energy_test))
print ("Качество ансамбля, 20 зданий:",
       energy_test_lr_rmsle, round(energy_test_lr_rmsle, 1))
print ("Качество ансамбля суммы, 20 зданий:",
       energy_test_sum_rmsle, round(energy_test_sum_rmsle, 1))
```

```
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:6: RuntimeWarning: ov
erflow encountered in exp

Качество ансамбля, 20 зданий: 0.2094817151521595 0.2
Качество ансамбля суммы, 20 зданий: 0.19320450684441387 0.2
```

РАСЧЕТ РЕЗУЛЬТАТОВ

Постановка задачи

Посчитаем модели линейной регрессии для 20 зданий по оптимальному набору параметров: метеорологические данные, дни недели, недели года, месяцы и праздники по всему набору данных.

Загрузим данные решения, посчитаем значение энергопотребления для требуемых дат для тех зданий, которые посчитаны в модели, и выгрузим результат в виде файла.

Данные:

1. http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz
2. http://video.ittensive.com/machine-learning/ashrae/weather_train.csv.gz
3. http://video.ittensive.com/machine-learning/ashrae/train.0.csv.gz
4. http://video.ittensive.com/machine-learning/ashrae/test.csv.gz
5. http://video.ittensive.com/machine-learning/ashrae/weather_test.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
import numpy as np
from scipy.interpolate import interp1d
from sklearn.linear_model import LinearRegression
```

Загрузка данных 20 зданий из HDF5

Ввод [2]:
```
energy = pd.read_hdf('energy.0-20.ready.h5', "energy")
print (energy.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 175680 entries, 0 to 175679
Data columns (total 92 columns):
timestamp                175680 non-null datetime64[ns]
building_id              175680 non-null int8
meter_reading            175680 non-null float16
primary_use              175680 non-null category
air_temperature          175680 non-null float16
cloud_coverage           175680 non-null float16
dew_temperature          175680 non-null float16
precip_depth_1_hr        175680 non-null float16
sea_level_pressure       175680 non-null float16
wind_direction           175680 non-null float16
wind_speed               175680 non-null float16
hour                     175680 non-null int8
weekday                  175680 non-null int8
week                     175680 non-null int8
month                    175680 non-null int8
date                     175680 non-null category
is_holiday               175680 non-null int8
is_wday0                 175680 non-null int8
is_wday1                 175680 non-null int8
is_wday2                 175680 non-null int8
is_wday3                 175680 non-null int8
is_wday4                 175680 non-null int8
is_wday5                 175680 non-null int8
is_wday6                 175680 non-null int8
is_w1                    175680 non-null int8
is_w2                    175680 non-null int8
is_w3                    175680 non-null int8
is_w4                    175680 non-null int8
is_w5                    175680 non-null int8
is_w6                    175680 non-null int8
is_w7                    175680 non-null int8
is_w8                    175680 non-null int8
is_w9                    175680 non-null int8
is_w10                   175680 non-null int8
is_w11                   175680 non-null int8
is_w12                   175680 non-null int8
is_w13                   175680 non-null int8
is_w14                   175680 non-null int8
is_w15                   175680 non-null int8
is_w16                   175680 non-null int8
is_w17                   175680 non-null int8
is_w18                   175680 non-null int8
is_w19                   175680 non-null int8
is_w20                   175680 non-null int8
```

```
is_w21                    175680 non-null int8
is_w22                    175680 non-null int8
is_w23                    175680 non-null int8
is_w24                    175680 non-null int8
is_w25                    175680 non-null int8
is_w26                    175680 non-null int8
is_w27                    175680 non-null int8
is_w28                    175680 non-null int8
is_w29                    175680 non-null int8
is_w30                    175680 non-null int8
is_w31                    175680 non-null int8
is_w32                    175680 non-null int8
is_w33                    175680 non-null int8
is_w34                    175680 non-null int8
is_w35                    175680 non-null int8
is_w36                    175680 non-null int8
is_w37                    175680 non-null int8
is_w38                    175680 non-null int8
is_w39                    175680 non-null int8
is_w40                    175680 non-null int8
is_w41                    175680 non-null int8
is_w42                    175680 non-null int8
is_w43                    175680 non-null int8
is_w44                    175680 non-null int8
is_w45                    175680 non-null int8
is_w46                    175680 non-null int8
is_w47                    175680 non-null int8
is_w48                    175680 non-null int8
is_w49                    175680 non-null int8
is_w50                    175680 non-null int8
is_w51                    175680 non-null int8
is_w52                    175680 non-null int8
is_w53                    175680 non-null int8
is_m1                     175680 non-null int8
is_m2                     175680 non-null int8
is_m3                     175680 non-null int8
is_m4                     175680 non-null int8
is_m5                     175680 non-null int8
is_m6                     175680 non-null int8
is_m7                     175680 non-null int8
is_m8                     175680 non-null int8
is_m9                     175680 non-null int8
is_m10                    175680 non-null int8
is_m11                    175680 non-null int8
is_m12                    175680 non-null int8
air_temperature_diff1     175680 non-null float16
air_temperature_diff2     175680 non-null float16
meter_reading_log         175680 non-null float16
dtypes: category(2), datetime64[ns](1), float16(11), int8(78)
memory usage: 20.0 MB
None
```

Загрузка данных для расчета, оптимизация памяти

Ввод [3]:
```python
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
```

```
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        elif col == "timestamp":
            df[col] = pd.to_datetime(df[col])
        elif str(col_type)[:8] != "datetime":
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                          / start_mem, 1), '%)')
    return df
```

Все результаты в оперативной памяти занимают порядка 8 Гб. Для оптимизации потребления памяти сначала рассчитаем результаты только для первых 20 зданий, а затем присоединим к ним остальные, заполненные нулями.

Ввод [6]:
```
buildings = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/building_metadata.csv.gz",
                        usecols=["site_id", "building_id"])
weather = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/weather_test.csv.gz")
weather = weather[weather["site_id"] == 0]
weather = weather.drop(columns=["wind_direction"], axis=1)
results = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/test.csv.gz")
results = results[(results["building_id"] < 20) & (results["meter"] == 0)]
results = pd.merge(left=results, right=buildings, how="left",
                   left_on="building_id", right_on="building_id")
del buildings
results = results.drop(columns=["meter"], axis=1)
print (results.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 350400 entries, 0 to 350399
Data columns (total 4 columns):
row_id         350400 non-null int64
building_id    350400 non-null int64
timestamp      350400 non-null object
site_id        350400 non-null int64
dtypes: int64(3), object(1)
memory usage: 13.4+ MB
None
```

Интерполяция значений и обогащение погодных данных: только для 1 города

Ввод [7]:
```
interpolate_columns = ["air_temperature", "dew_temperature",
                       "cloud_coverage", "wind_speed",
                       "sea_level_pressure"]
for col in interpolate_columns:
    weather[col] = weather[col].interpolate(limit_direction='both',
                                kind='cubic')
weather["air_temperature_diff1"] = weather["air_temperature"].diff()
weather.at[0, "air_temperature_diff1"] = weather.at[1, "air_temperature_diff1"]
weather["air_temperature_diff2"] = weather["air_temperature_diff1"].diff()
weather.at[0, "air_temperature_diff2"] = weather.at[1, "air_temperature_diff2"]
```

Объединение данных по погоде

Ввод [8]:
```
results = results.set_index(["timestamp", "site_id"])
weather = weather.set_index(["timestamp", "site_id"])
results = pd.merge(left=results, right=weather, how="left",
                   left_index=True, right_index=True)
```

```
results.reset_index(inplace=True)
results = results.drop(columns=["site_id"], axis=1)
del weather
results = reduce_mem_usage(results)
print (results.info())
```

```
Потребление памяти меньше на 19.72 Мб (минус 67.0 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 350400 entries, 0 to 350399
Data columns (total 11 columns):
timestamp                 350400 non-null datetime64[ns]
row_id                    350400 non-null int32
building_id               350400 non-null int8
air_temperature           350400 non-null float16
cloud_coverage            350400 non-null float16
dew_temperature           350400 non-null float16
precip_depth_1_hr         350400 non-null float16
sea_level_pressure        350400 non-null float16
wind_speed                350400 non-null float16
air_temperature_diff1     350400 non-null float16
air_temperature_diff2     350400 non-null float16
dtypes: datetime64[ns](1), float16(8), int32(1), int8(1)
memory usage: 9.7 MB
None
```

Обогащение данных по дате

Ввод [9]:
```
results["hour"] = results["timestamp"].dt.hour.astype("int8")
results["weekday"] = results["timestamp"].dt.weekday.astype("int8")
results["week"] = results["timestamp"].dt.week.astype("int8")
results["month"] = results["timestamp"].dt.month.astype("int8")
results["date"] = pd.to_datetime(energy["timestamp"].dt.date)
dates_range = pd.date_range(start='2016-12-31', end='2018-06-01')
us_holidays = calendar().holidays(start=dates_range.min(),
                                  end=dates_range.max())
results['is_holiday'] = results['date'].isin(us_holidays).astype("int8")
for weekday in range(0,7):
    results['is_wday' + str(weekday)] = results['weekday'].isin([weekday]).astype("int8")
for week in range(1,54):
    results['is_w' + str(week)] = results['week'].isin([week]).astype("int8")
for month in range(1,13):
    results['is_m' + str(month)] = results['month'].isin([month]).astype("int8")
```

Линейная регрессия

Ввод [10]:
```
hours = range(0, 24)
buildings = range(0, energy["building_id"].max() + 1)
lr_columns = ["meter_reading_log", "hour", "building_id",
              "air_temperature", "dew_temperature",
              "sea_level_pressure", "wind_speed", "cloud_coverage",
              "air_temperature_diff1", "air_temperature_diff2",
              "is_holiday"]
for wday in range(0,7):
    lr_columns.append("is_wday" + str(wday))
for week in range(1,54):
    lr_columns.append("is_w" + str(week))
for month in range(1,13):
    lr_columns.append("is_m" + str(month))
energy_train_lr = pd.DataFrame(energy, columns=lr_columns)
energy_lr = [[]]*len(buildings)
for building in buildings:
    energy_lr[building] = [[]]*len(hours)
    energy_train_b = energy_train_lr[energy_train_lr["building_id"]==building]
    for hour in hours:
        energy_train_bh = energy_train_b[energy_train_b["hour"]==hour]
        y = energy_train_bh["meter_reading_log"]
        x = energy_train_bh.drop(labels=["meter_reading_log",
            "hour", "building_id"], axis=1)
```

```
        model = LinearRegression(fit_intercept=False).fit(x, y)
        energy_lr[building][hour] = model.coef_
        energy_lr[building][hour] = np.append(energy_lr[building][hour], model.intercept_)
print (energy_lr[0])
```

```
[array([ 1.88484471e-02, -4.35195165e-03, -2.43625045e-03, -2.11815834e-02,
       -1.61005408e-02,  5.26762009e-03,  5.82905114e-03,  2.33321190e-02,
        3.24352765e+00,  3.26037359e+00,  3.27985382e+00,  3.24306774e+00,
        3.25154567e+00,  3.35423684e+00,  3.33826852e+00, -1.06680721e-01,
       -1.41408086e-01, -1.20692968e-01, -1.27921462e-01, -1.83425403e+00,
       -1.80152631e+00, -1.84464943e+00, -1.81480956e+00, -1.84290886e+00,
       -1.82876682e+00, -1.88813031e+00, -1.79642034e+00, -1.81421185e+00,
       -1.76295269e+00, -1.68784475e+00, -1.72124350e+00, -1.73937881e+00,
       -1.87428641e+00, -1.86946774e+00, -3.00204873e-01,  3.41759872e+00,
        3.25210738e+00,  3.34519267e+00,  3.40270352e+00,  3.25671077e+00,
        3.27047157e+00,  3.16922760e+00,  3.38226604e+00,  3.38442254e+00,
        3.32446218e+00,  9.11405087e-01,  1.01003397e+00,  1.00636315e+00,
        8.45111668e-01,  8.28644621e-01,  7.91694760e-01,  7.76295543e-01,
        7.54676163e-01,  8.59878063e-01,  9.43091452e-01,  9.73485649e-01,
        8.82133901e-01,  8.19480777e-01,  8.01830351e-01,  7.45418668e-01,
        7.03975022e-01,  8.97737026e-01,  7.57105231e-01,  3.90105158e-01,
        5.10084987e-01,  8.24575961e-01,  9.02695894e-01, -2.53619909e-01,
       -7.50287533e-01,  9.07731056e-01,  8.68673205e-01,  7.47440338e-01,
        8.30523491e-01,  1.17579079e+00,  1.25148630e+00,  3.64982009e+00,
        3.69203591e+00,  3.54740906e+00,  3.52447104e+00,  3.52509689e+00,
        0.00000000e+00]), array([ 2.84966081e-02, -9.51369572e-03, -3.93086672e-03, -3.03342938e-02,
       -1.49178170e-02,  8.65143538e-03,  6.73317909e-03,  4.12211418e-02,
        4.06474972e+00,  4.09233809e+00,  4.10482121e+00,  4.08304310e+00,
        4.07178450e+00,  4.18821764e+00,  4.18564415e+00,  1.75486803e-02,
       -8.34071636e-03,  3.59773636e-03, -3.51948142e-02, -1.82555938e+00,
       -1.80223584e+00, -1.87971628e+00, -1.87149715e+00, -1.80013180e+00,
       -1.79387283e+00, -1.86853635e+00, -1.74373245e+00, -1.80546880e+00,
       -1.76129603e+00, -1.66324627e+00, -1.69587684e+00, -1.74283779e+00,
       -1.70440531e+00, -1.73508358e+00, -1.19023383e-01,  3.56397748e+00,
        3.50920773e+00,  3.60184360e+00,  3.63464570e+00,  3.51408529e+00,
        3.47969294e+00,  3.39310670e+00,  3.61000490e+00,  3.60910678e+00,
        3.54474854e+00,  1.02924871e+00,  1.06340325e+00,  1.10539711e+00,
        9.17617559e-01,  8.92487407e-01,  9.15935636e-01,  8.98509026e-01,
        8.65895748e-01,  9.81914282e-01,  1.03640461e+00,  1.10421014e+00,
        9.39942241e-01,  8.65573227e-01,  8.96537840e-01,  8.38786960e-01,
        8.12616885e-01,  9.97735381e-01,  8.72648835e-01,  5.72924495e-01,
        6.63927317e-01,  1.00829637e+00,  1.07325006e+00, -1.88814521e-01,
       -2.13778496e-01,  1.55001378e+00,  1.43060291e+00,  1.30839944e+00,
        1.25614429e+00,  1.51419592e+00,  1.58597279e+00,  4.11466265e+00,
        4.13183260e+00,  4.09363031e+00,  4.02970076e+00,  3.98009491e+00,
        0.00000000e+00]), array([ 4.53025214e-02, -2.14493349e-02, -2.14573741e-03, -1.64947286e-02,
       -1.70259774e-02,  1.74464285e-02,  1.97693110e-02,  1.74393654e-02,
        2.91668630e+00,  2.92261577e+00,  2.93829823e+00,  2.92539191e+00,
        2.91708088e+00,  3.01092339e+00,  3.01283669e+00, -1.06493711e-01,
       -1.86239511e-01, -1.27257347e-01, -1.49863720e-01, -1.81373227e+00,
       -1.84607863e+00, -1.91076779e+00, -1.91750574e+00, -1.93401313e+00,
       -1.99560499e+00, -2.05656648e+00, -1.91419601e+00, -1.97541440e+00,
       -1.87068045e+00, -1.79021263e+00, -1.80318713e+00, -1.87209439e+00,
       -1.91031265e+00, -1.99867678e+00, -3.36347967e-01,  3.33274817e+00,
        3.29804778e+00,  3.43385220e+00,  3.49410772e+00,  3.26379156e+00,
        3.36495829e+00,  3.16114497e+00,  3.41634703e+00,  3.36683893e+00,
        3.35487413e+00,  1.13318014e+00,  1.18743896e+00,  1.17759955e+00,
        9.87378061e-01,  1.02615178e+00,  1.00420845e+00,  9.90437627e-01,
        9.80704784e-01,  1.02254558e+00,  9.24851418e-01,  9.64962482e-01,
        8.36965382e-01,  7.74091184e-01,  5.47416210e-01,  3.96122396e-01,
        4.21564579e-01,  5.49250185e-01,  4.63944972e-01,  1.17652178e-01,
        2.46580482e-01,  5.59945464e-01,  6.53295875e-01, -2.93089867e-01,
       -8.65218937e-01,  7.65448570e-01,  7.67410994e-01,  5.60221553e-01,
        6.59416735e-01,  8.36508632e-01,  9.54116821e-01,  3.20136642e+00,
        3.20896769e+00,  3.33290267e+00,  3.60616016e+00,  3.61820984e+00,
        0.00000000e+00]), array([ 5.40797040e-02, -2.65859365e-02, -2.65865028e-03, -2.15999335e-02,
       -1.42651498e-02, -8.88496637e-03, -1.23442411e-02, -1.93886757e-02,
        3.16623521e+00,  3.15003967e+00,  3.18187428e+00,  3.15204978e+00,
        3.16470647e+00,  3.25772572e+00,  3.25655365e+00, -7.80088305e-02,
       -1.55607194e-01, -8.50646496e-02, -1.09641954e-01, -1.95058441e+00,
       -1.94962764e+00, -2.01507592e+00, -2.03545594e+00, -1.99189007e+00,
       -2.00848055e+00, -2.09138536e+00, -1.94335091e+00, -1.98022425e+00,
       -1.85382080e+00, -1.76767075e+00, -1.76102185e+00, -1.85310650e+00,
       -1.86653900e+00, -1.96152163e+00, -2.46601760e-01,  3.38502645e+00,
        3.40706205e+00,  3.56875372e+00,  3.58257961e+00,  3.36524081e+00,
        3.48837233e+00,  3.29296017e+00,  3.55284309e+00,  3.50780725e+00,
        3.46190047e+00,  1.08064067e+00,  1.06829321e+00,  1.09313571e+00,
        8.97076011e-01,  9.70919251e-01,  9.48685288e-01,  9.21699405e-01,
        8.99895251e-01,  1.00166118e+00,  9.72638488e-01,  1.03912187e+00,
        8.88599515e-01,  8.54711890e-01,  6.92933917e-01,  5.99670172e-01,
        5.05302191e-01,  6.35917425e-01,  5.97590327e-01,  2.46748090e-01,
```

```
 3.20205033e-01,  6.90438032e-01,  7.69840956e-01, -2.73228586e-01,
-7.01243401e-01,  1.07598901e+00,  9.83403386e-01,  7.21868992e-01,
 7.41525650e-01,  8.87157559e-01,  9.23165798e-01,  3.44280362e+00,
 3.44508219e+00,  3.44526649e+00,  3.65688396e+00,  3.70637894e+00,
 0.00000000e+00]), array([ 5.93091436e-02, -3.00981458e-02, -9.80108976e-04,  6.72703981e-03,
-1.10926628e-02, -8.54367018e-03, -5.65552711e-03, -4.59995270e-02,
 2.09287596e+00,  2.06157064e+00,  2.12288141e+00,  2.10649014e+00,
 2.10500693e+00,  2.18885994e+00,  2.22252226e+00, -2.48818085e-01,
-2.89007038e-01, -2.17481613e-01, -2.22221047e-01, -2.21361375e+00,
-2.17640924e+00, -2.21989107e+00, -2.22508192e+00, -2.15638900e+00,
-2.20968556e+00, -2.26133776e+00, -2.13130522e+00, -2.17517495e+00,
-2.04089165e+00, -2.01174116e+00, -2.00508904e+00, -2.04238153e+00,
-2.04616714e+00, -2.13082385e+00, -4.63024139e-01,  3.19211030e+00,
 3.32788277e+00,  3.48268485e+00,  3.54406071e+00,  3.28679395e+00,
 3.46753454e+00,  3.29326463e+00,  3.55548692e+00,  3.52574682e+00,
 3.48246145e+00,  8.30392003e-01,  7.68671811e-01,  8.42731416e-01,
 6.35378361e-01,  6.57663226e-01,  6.71257854e-01,  6.28298402e-01,
 6.71731710e-01,  7.30504811e-01,  6.33410931e-01,  7.36296117e-01,
 6.26352549e-01,  5.34748197e-01,  6.41672134e-01,  6.49538517e-01,
 5.86439490e-01,  7.12867916e-01,  6.43707156e-01,  3.20630014e-01,
 4.25354242e-01,  7.82008708e-01,  8.71227443e-01, -3.69745374e-01,
-1.34595799e+00,  5.36894321e-01,  3.85068178e-01,  1.95248127e-01,
 1.68162704e-01,  2.03158975e-01,  1.60964489e-01,  2.92436409e+00,
 2.96621561e+00,  2.93594408e+00,  2.92762637e+00,  2.84488344e+00,
 0.00000000e+00]), array([ 6.15367666e-02, -3.33100073e-02,  2.58590281e-03, -4.58163023e-03,
-1.50680542e-02, -2.35261917e-02, -2.33457088e-02, -1.96237564e-02,
 1.98112726e-02,  8.41832161e-03,  5.24012148e-02,  4.51814011e-03,
 1.70720220e-02,  1.08968258e-01,  1.70929074e-01, -4.90711778e-01,
-5.73917627e-01, -5.81732035e-01, -3.92711163e-03, -2.35097313e+00,
-2.41515183e+00, -2.39958429e+00, -2.43524408e+00, -2.48557425e+00,
-2.53073621e+00, -2.45483780e+00, -2.40707660e+00, -2.42074442e+00,
-2.40427828e+00, -2.32820559e+00, -2.34608173e+00, -2.40189981e+00,
-2.22452593e+00, -2.32268262e+00, -6.15109622e-01,  3.01320052e+00,
 3.05214596e+00,  3.16732192e+00,  3.23412132e+00,  2.93612194e+00,
 3.13102603e+00,  2.96021700e+00,  3.18471503e+00,  3.14884305e+00,
 3.12771559e+00,  1.88979506e-02,  5.17960787e-02,  3.11660171e-02,
-8.42599869e-02,  1.21179461e-01,  1.34994030e-01,  1.07663274e-01,
 1.66156530e-01,  2.92943299e-01,  5.39758086e-01,  5.30017555e-01,
 3.39320362e-01,  3.12847972e-01,  6.31244302e-01,  6.55158699e-01,
 5.81417143e-01,  7.08169639e-01,  6.74088597e-01,  3.96747619e-01,
 4.65178967e-01,  8.55203986e-01,  9.57994640e-01, -8.68447125e-01,
-2.51864719e+00, -7.53584266e-01, -8.20170641e-01, -9.49087858e-01,
-1.14063954e+00, -9.25256491e-01, -9.99185562e-01,  2.16749811e+00,
 1.99541187e+00,  1.70597255e+00,  1.33913851e+00,  1.28063703e+00,
 0.00000000e+00]), array([ 4.27215658e-02, -1.79940667e-02, -9.22217965e-04, -6.60461187e-03,
-1.61936283e-02, -2.04877853e-02, -2.04186440e-02, -3.59883308e-02,
 2.20233750e+00,  2.16783285e+00,  2.19670582e+00,  2.21040010e+00,
 2.18496847e+00,  2.28120637e+00,  2.31362724e+00, -2.49702692e-01,
-2.57577121e-01, -2.24398077e-01, -2.53621399e-01, -2.24510336e+00,
-2.19398904e+00, -2.25574541e+00, -2.22974372e+00, -2.12567949e+00,
-2.13727856e+00, -2.20177197e+00, -2.11003399e+00, -2.09685278e+00,
-1.86216784e+00, -1.88743508e+00, -1.87257886e+00, -1.97407842e+00,
-1.94087064e+00, -1.98999476e+00, -4.07420635e-01,  3.31517792e+00,
 3.27778840e+00,  3.40832996e+00,  3.48179674e+00,  3.28094506e+00,
 3.38175297e+00,  3.17920256e+00,  3.41010380e+00,  3.39752889e+00,
 3.38706398e+00,  7.88369417e-01,  7.57713974e-01,  8.55463266e-01,
 6.90246105e-01,  7.42878735e-01,  7.72420228e-01,  7.19788074e-01,
 7.18510151e-01,  8.29560995e-01,  7.81517982e-01,  8.57394457e-01,
 7.38995492e-01,  7.02076674e-01,  6.01764500e-01,  6.13717377e-01,
 5.28060257e-01,  6.69569373e-01,  5.27261734e-01,  1.96594536e-01,
 2.96224236e-01,  6.60686731e-01,  7.95256376e-01, -2.86824822e-01,
-1.27360320e+00,  6.22226238e-01,  3.85912418e-01,  1.75793409e-01,
 1.93834931e-01,  2.81223416e-01,  3.28754902e-01,  2.96985626e+00,
 2.95881724e+00,  2.90499187e+00,  3.01267242e+00,  2.99402785e+00,
 0.00000000e+00]), array([ 2.90760840e-03,  1.22640328e-02, -1.40848756e-03,  2.77506113e-02,
-1.81553364e-02, -2.22243965e-02, -1.65767670e-02, -7.92741776e-03,
 2.60085106e+00,  2.59121442e+00,  2.61410737e+00,  2.61128879e+00,
 2.59403563e+00,  2.71443820e+00,  2.70280027e+00, -2.24477351e-01,
-1.66714132e-01, -1.87852979e-01, -1.63796142e-01, -2.16753054e+00,
-2.10490108e+00, -2.16628265e+00, -2.15585542e+00, -2.03798342e+00,
-2.03918839e+00, -2.05485058e+00, -1.97172952e+00, -2.02774239e+00,
-1.90219343e+00, -1.88674068e+00, -1.93627596e+00, -1.97948253e+00,
-1.88085556e+00, -1.87855339e+00, -2.99113035e-01,  3.48915339e+00,
 3.26722312e+00,  3.37270641e+00,  3.47355127e+00,  3.30981159e+00,
 3.35677123e+00,  3.30130005e+00,  3.52479196e+00,  3.49769020e+00,
 3.43009353e+00,  6.31217778e-01,  6.83552563e-01,  6.94734335e-01,
 5.53924799e-01,  5.66466808e-01,  5.67522407e-01,  5.12713552e-01,
 6.32760406e-01,  6.63830519e-01,  7.32148170e-01,  8.00972700e-01,
 6.41008019e-01,  6.59186542e-01,  8.08457196e-01,  8.37833166e-01,
 7.24934101e-01,  8.92887473e-01,  8.07900012e-01,  6.49304986e-01,
 7.39699125e-01,  1.06364977e+00,  1.18741941e+00, -4.07117724e-01,
-1.14722729e+00,  7.69886971e-01,  5.88494778e-01,  4.72867012e-01,
 3.97037983e-01,  6.65303528e-01,  6.37050867e-01,  3.46171713e+00,
 3.42584920e+00,  3.24685502e+00,  3.06992531e+00,  2.84574842e+00,
```

```
 0.00000000e+00]), array([ 3.92386829e-03,  1.08773774e-03, -6.58339262e-03, -2.43949890e-03,
-1.39167309e-02, -3.71618271e-02, -3.89716625e-02,  6.98263645e-02,
 5.76105356e+00,  5.78391409e+00,  5.79229164e+00,  5.79068375e+00,
 5.81692982e+00,  6.04726219e+00,  5.83197832e+00,  1.14736751e-01,
 1.15109771e-01,  7.34225512e-02,  6.31762266e-01, -1.91489589e+00,
-1.94955564e+00, -1.93863153e+00, -1.97042155e+00, -1.73298442e+00,
-1.65260744e+00, -1.64541471e+00, -1.64618039e+00, -1.57880497e+00,
-1.54508734e+00, -1.53074610e+00, -1.56658018e+00, -1.57711554e+00,
-1.45565057e+00, -1.38667154e+00,  2.60921240e-01,  3.92700911e+00,
 3.62961626e+00,  3.77902317e+00,  3.86093116e+00,  3.83554792e+00,
 3.85016394e+00,  3.92912507e+00,  4.25892591e+00,  4.21952152e+00,
 4.18236780e+00,  9.64523792e-01,  9.92247820e-01,  9.49122787e-01,
 7.35413551e-01,  8.99374366e-01,  9.86858487e-01,  9.59884167e-01,
 1.04052162e+00,  1.05310309e+00,  1.37229490e+00,  1.49845314e+00,
 1.29535663e+00,  1.22803819e+00,  1.36276758e+00,  1.33117878e+00,
 1.34927797e+00,  1.30393338e+00,  1.32690597e+00,  9.44081664e-01,
 1.05655372e+00,  1.39819646e+00,  1.32095742e+00, -1.23619080e-01,
 8.15499783e-01,  2.75687075e+00,  2.48728704e+00,  2.33857822e+00,
 2.23989868e+00,  2.53862286e+00,  2.28656459e+00,  5.55816841e+00,
 5.28743172e+00,  4.91212368e+00,  4.77531385e+00,  4.83286476e+00,
 0.00000000e+00]), array([ 1.61955096e-02, -9.52816755e-03, -7.42529333e-03, -1.88800693e-02,
-1.00396872e-02,  3.01382217e+01, -3.01200676e+01,  1.13777637e-01,
 6.27282143e+00,  6.28110504e+00,  6.30779696e+00,  6.26054001e+00,
 6.23170900e+00,  6.31226110e+00,  6.41183805e+00,  2.59212673e-01,
 2.99598098e-01,  2.02491045e-01,  2.60698795e-01, -1.47156703e+00,
-1.47956586e+00, -1.51914907e+00, -1.49809694e+00, -1.54198146e+00,
-1.56963897e+00, -1.58100390e+00, -1.55631316e+00, -1.56631994e+00,
-1.52501583e+00, -1.49201393e+00, -1.48890007e+00, -1.56991529e+00,
-1.53477001e+00, -1.48600340e+00,  1.86939478e-01,  3.68664074e+00,
 3.65794659e+00,  3.82901287e+00,  3.95830679e+00,  3.86386847e+00,
 3.83396935e+00,  3.85813403e+00,  4.11718225e+00,  4.11570883e+00,
 4.08381557e+00,  1.20228791e+00,  1.23594284e+00,  1.19029212e+00,
 1.02285075e+00,  1.09715128e+00,  1.04326129e+00,  1.09676790e+00,
 1.03177583e+00,  1.16294634e+00,  1.39956999e+00,  1.43550611e+00,
 1.38198173e+00,  1.29094887e+00,  1.28167033e+00,  1.29495180e+00,
 1.36407208e+00,  1.35992956e+00,  1.28748274e+00,  9.56931829e-01,
 1.01704431e+00,  1.37939572e+00,  1.34390807e+00,  8.44106674e-02,
 1.03576088e+00,  2.78873038e+00,  2.69308043e+00,  2.71411228e+00,
 2.76979160e+00,  2.75063419e+00,  2.62006474e+00,  5.68112469e+00,
 5.55990267e+00,  5.37858009e+00,  5.12681293e+00,  5.11046505e+00,
 0.00000000e+00]), array([ 2.23711338e-02, -1.40241627e-02, -6.78506494e-03, -1.52268410e-02,
-1.25137568e-02,  2.45467796e+01, -2.45261421e+01,  4.92153168e-02,
 5.83786678e+00,  5.88925362e+00,  5.88940239e+00,  5.86317015e+00,
 5.88943338e+00,  5.99432707e+00,  5.96751690e+00,  1.68501139e-01,
 1.88651919e-01,  1.87330842e-01,  1.86014295e-01, -1.70894766e+00,
-1.73480535e+00, -1.72206748e+00, -1.74212861e+00, -1.66797125e+00,
-1.64513028e+00, -1.68739104e+00, -1.64362288e+00, -1.62819457e+00,
-1.55477655e+00, -1.55710375e+00, -1.52940190e+00, -1.57885170e+00,
-1.46164846e+00, -1.45222354e+00,  2.04694152e-01,  3.88742685e+00,
 3.73372269e+00,  3.86781287e+00,  3.99036622e+00,  3.91676211e+00,
 3.83446383e+00,  3.79021573e+00,  4.02488995e+00,  4.00856400e+00,
 3.97182560e+00,  1.28096056e+00,  1.30307198e+00,  1.28226328e+00,
 1.12307024e+00,  1.11333847e+00,  1.10011601e+00,  1.09181523e+00,
 1.07238173e+00,  1.15805030e+00,  1.34478092e+00,  1.43597305e+00,
 1.36227810e+00,  1.15977478e+00,  1.07822466e+00,  1.10839891e+00,
 1.22643828e+00,  1.21333933e+00,  1.11162424e+00,  7.46216774e-01,
 8.85584831e-01,  1.24050903e+00,  1.15005279e+00,  8.49816799e-02,
 8.20567131e-01,  2.68811226e+00,  2.57801437e+00,  2.43743229e+00,
 2.32976532e+00,  2.48421621e+00,  2.50255108e+00,  5.26418352e+00,
 5.20981789e+00,  4.99439859e+00,  5.01531410e+00,  5.01710129e+00,
 0.00000000e+00]), array([ 1.78020373e-02, -1.41881276e-02, -1.31571293e-02, -2.12068558e-02,
-3.41033936e-03, -1.31382265e+01,  1.31122046e+01,  5.23209572e-02,
 9.69803238e+00,  9.69894600e+00,  9.72238731e+00,  9.69177341e+00,
 9.69162941e+00,  9.76525116e+00,  9.80967426e+00,  6.29802525e-01,
 5.78488529e-01,  6.53383255e-01,  6.34139061e-01, -1.21704769e+00,
-1.22093093e+00, -1.22314787e+00, -1.23844552e+00, -1.20192075e+00,
-1.14520252e+00, -1.24894667e+00, -1.18679893e+00, -1.17745352e+00,
-1.12007475e+00, -1.08697701e+00, -1.11532545e+00, -1.14570022e+00,
-1.08205795e+00, -1.03199315e+00,  6.05084896e-01,  4.33175898e+00,
 4.10936928e+00,  4.45855474e+00,  4.63533926e+00,  4.55124664e+00,
 4.48070240e+00,  4.49599695e+00,  4.70446491e+00,  4.71335793e+00,
 4.67615938e+00,  1.87169218e+00,  1.89681149e+00,  1.92535162e+00,
 1.75109482e+00,  1.69533658e+00,  1.72808504e+00,  1.71385312e+00,
 1.68494177e+00,  1.71787524e+00,  1.78278399e+00,  1.91594100e+00,
 1.85125589e+00,  1.63076925e+00,  1.54103756e+00,  1.49122751e+00,
 1.59593785e+00,  1.59138274e+00,  1.53949893e+00,  1.17603683e+00,
 1.31632400e+00,  1.69208944e+00,  1.57379961e+00,  5.60527802e-01,
 3.05350113e+00,  4.90881157e+00,  4.81637049e+00,  4.74934626e+00,
 4.63919115e+00,  4.60572338e+00,  4.59431791e+00,  7.42060995e+00,
 7.38010168e+00,  7.26309013e+00,  7.30529213e+00,  7.33253002e+00,
 0.00000000e+00]), array([ 3.51328179e-02, -2.79576741e-02, -1.71235949e-02, -4.49877977e-02,
-8.75848532e-03,  1.11031532e-03, -2.40385532e-02,  5.45475483e-02,
 1.20573015e+01,  1.19278755e+01,  1.20689735e+01,  1.19394970e+01,
 1.21461859e+01,  1.21939888e+01,  1.21839094e+01,  9.80486780e-01,
```

```
 9.29427147e-01,  1.01141858e+00,  9.10523534e-01, -4.85273480e-01,
-5.28448582e-01, -5.05071521e-01, -5.77165127e-01, -6.91798329e-01,
-6.03195548e-01, -7.31099188e-01, -6.32640600e-01, -6.37228370e-01,
-6.19913578e-01, -5.89356899e-01, -5.49468040e-01, -6.59235954e-01,
-8.06097031e-01, -6.79471374e-01,  8.85809541e-01,  4.59243631e+00,
 4.13847876e+00,  4.30191088e+00,  4.50955915e+00,  4.46451092e+00,
 4.39649248e+00,  4.52180004e+00,  4.68756056e+00,  4.65911245e+00,
 4.60676193e+00,  1.79245317e+00,  1.75229168e+00,  1.76024234e+00,
 1.61845005e+00,  1.68553782e+00,  1.78799200e+00,  1.72195768e+00,
 1.66444075e+00,  1.70151877e+00,  1.80244768e+00,  1.99945009e+00,
 1.86000037e+00,  1.74342179e+00,  2.12293601e+00,  2.15040112e+00,
 2.10031533e+00,  2.28127718e+00,  2.31770015e+00,  2.20139503e+00,
 2.34850979e+00,  2.71460819e+00,  2.41986322e+00,  6.92778111e-01,
 4.52076960e+00,  5.85928059e+00,  5.99689627e+00,  5.94093800e+00,
 6.00809097e+00,  6.37131977e+00,  6.30126143e+00,  9.22979450e+00,
 9.02470303e+00,  8.87934208e+00,  8.36326313e+00,  8.05711079e+00,
 0.00000000e+00]), array([ 2.37652380e-02, -1.62699558e-02, -1.31657124e-02, -2.14029253e-02,
-6.72841072e-03, -3.29720974e-03, -9.28997993e-03,  1.43906116e-01,
 9.62320995e+00,  9.68727970e+00,  9.68492031e+00,  9.66081619e+00,
 9.66494751e+00,  9.79842567e+00,  9.77445602e+00,  6.76542640e-01,
 6.32934093e-01,  7.02080727e-01,  6.53622448e-01, -6.29606843e-01,
-6.73082888e-01, -6.87969089e-01, -7.01029003e-01, -9.23200130e-01,
-8.95810843e-01, -9.82383728e-01, -9.52364683e-01, -9.09164429e-01,
-8.72882843e-01, -8.77165318e-01, -8.91857147e-01, -9.71760035e-01,
-1.15013790e+00, -1.09788465e+00,  5.78939199e-01,  4.22931910e+00,
 3.85248995e+00,  3.92330050e+00,  4.03243637e+00,  3.97641110e+00,
 3.97974491e+00,  4.11496115e+00,  4.29787302e+00,  4.30156755e+00,
 4.24636078e+00,  1.34001434e+00,  1.39554310e+00,  1.35152435e+00,
 1.19803333e+00,  1.09086847e+00,  9.99483109e-01,  1.04245543e+00,
 9.90726709e-01,  1.16324091e+00,  1.60538340e+00,  1.68150294e+00,
 1.60859990e+00,  1.40644431e+00,  1.93456125e+00,  1.93289077e+00,
 2.02760983e+00,  2.00557041e+00,  2.26833057e+00,  2.11850119e+00,
 2.33442307e+00,  2.63381195e+00,  2.46406937e+00,  3.26587200e-01,
 2.99121761e+00,  4.46662712e+00,  4.59469509e+00,  4.44448042e+00,
 4.65756512e+00,  5.13503695e+00,  5.00804472e+00,  7.96099949e+00,
 7.92696905e+00,  7.46970081e+00,  6.83243799e+00,  6.39181423e+00,
 0.00000000e+00]), array([ 1.69383381e-02, -7.93357566e-03, -2.15870291e-02, -1.21910274e-02,
-1.81955397e-02,  8.10748339e-03,  7.22408295e-03,  4.01840210e-02,
 1.46401081e+01,  1.46670370e+01,  1.46822195e+01,  1.46290112e+01,
 1.47050924e+01,  1.47175188e+01,  1.47294216e+01,  1.24891877e+00,
 1.17355514e+00,  1.29646397e+00,  1.22965240e+00, -2.06546307e-01,
-2.03909397e-01, -2.30943441e-01, -2.46417284e-01, -3.64809871e-01,
-2.85840750e-01, -4.07768250e-01, -3.11036587e-01, -3.75946462e-01,
 2.42250204e-01, -3.30779076e-01, -3.58926296e-01, -4.44867611e-01,
-4.79432583e-01, -3.69895935e-01,  1.27614975e+00,  4.93927765e+00,
 4.61849880e+00,  4.72211885e+00,  4.79121113e+00,  4.79879570e+00,
 4.77448416e+00,  4.94185734e+00,  5.08314371e+00,  5.08692360e+00,
 5.01787186e+00,  2.54128551e+00,  2.58070850e+00,  2.56756639e+00,
 2.40623403e+00,  2.28781366e+00,  2.26305437e+00,  2.26758051e+00,
 2.21020198e+00,  2.25875115e+00,  2.29370379e+00,  2.49319601e+00,
 2.34809637e+00,  2.24052382e+00,  2.23541021e+00,  2.26401949e+00,
 2.27495909e+00,  2.30814385e+00,  2.23912859e+00,  1.96910095e+00,
 2.20321083e+00,  2.50484037e+00,  2.33640432e+00,  1.13931274e+00,
 6.06049156e+00,  7.50155735e+00,  7.53548574e+00,  7.48650169e+00,
 7.50682688e+00,  7.86592960e+00,  7.72615242e+00,  1.02558374e+01,
 1.03210506e+01,  1.02125187e+01,  1.01164989e+01,  1.01220913e+01,
 0.00000000e+00]), array([ 2.43349764e-02, -1.28744626e-02, -1.43232793e-02, -8.21590424e-03,
-2.40276456e-02, -4.63457108e-02, -4.66590524e-02,  4.07185555e-02,
 1.03650770e+01,  1.03988495e+01,  1.04072380e+01,  1.03648472e+01,
 1.03318415e+01,  1.04646406e+01,  1.04644156e+01,  6.14933848e-01,
 6.78364754e-01,  7.30314732e-01,  6.93741083e-01, -1.52712870e+00,
-1.48224115e+00, -1.49427128e+00, -1.58410597e+00, -8.79941046e-01,
-7.68067241e-01, -8.47104073e-01, -8.19387197e-01, -8.71258140e-01,
-7.53476381e-01, -7.52754986e-01, -7.66346633e-01, -8.12125087e-01,
-8.71847034e-01, -8.14976096e-01,  8.63629341e-01,  4.44309473e+00,
 4.39004469e+00,  4.42524624e+00,  4.46886444e+00,  4.55636644e+00,
 4.43567610e+00,  4.33723497e+00,  4.54227161e+00,  4.56836796e+00,
 4.53318977e+00,  2.26272249e+00,  2.28692198e+00,  2.21621895e+00,
 2.00283384e+00,  1.93096828e+00,  1.89385247e+00,  1.92190456e+00,
 1.90153313e+00,  1.92375541e+00,  1.80248666e+00,  2.00794101e+00,
 1.88577569e+00,  1.68063045e+00,  1.48761570e+00,  1.37454462e+00,
 1.47926748e+00,  1.53662896e+00,  1.47031748e+00,  1.05709958e+00,
 1.40536964e+00,  1.58801258e+00,  1.56917858e+00,  8.68538737e-01,
 3.58722854e+00,  5.70770121e+00,  5.04720306e+00,  4.81840324e+00,
 4.77446079e+00,  5.08252668e+00,  5.11518860e+00,  7.52771187e+00,
 7.54765081e+00,  7.71742725e+00,  7.92485094e+00,  7.97481298e+00,
 0.00000000e+00]), array([ 9.89645254e-03, -5.37309144e-03, -1.07008815e-02, -1.56992674e-03,
-1.66466534e-02, -8.05598497e-03, -1.70180798e-02,  4.77652550e-02,
 8.22455788e+00,  8.27179527e+00,  8.28828526e+00,  8.22991657e+00,
 8.22248840e+00,  8.33234119e+00,  8.31244659e+00,  4.49008226e-01,
 4.00445998e-01,  4.52516794e-01,  4.74473238e-01, -1.26668906e+00,
-1.27470124e+00, -1.26425219e+00, -1.30998397e+00, -1.24556148e+00,
-1.19124532e+00, -1.27526021e+00, -1.20239806e+00, -1.23080027e+00,
-1.18298459e+00, -1.15829158e+00, -1.16256881e+00, -1.22355962e+00,
```

```
-1.21175516e+00, -1.14662838e+00,  4.48859215e-01,  4.17302704e+00,
 3.96037102e+00,  4.01167965e+00,  3.99625468e+00,  4.10840559e+00,
 4.02360296e+00,  4.06888533e+00,  4.27643871e+00,  4.25963497e+00,
 4.25061131e+00,  1.75135756e+00,  1.77570391e+00,  1.76699662e+00,
 1.57590032e+00,  1.48584569e+00,  1.50401449e+00,  1.49109960e+00,
 1.50531363e+00,  1.50862193e+00,  1.52636909e+00,  1.64113736e+00,
 1.56481576e+00,  1.34875786e+00,  1.30388093e+00,  1.27831697e+00,
 1.34575558e+00,  1.35750365e+00,  1.34323907e+00,  9.95491982e-01,
 1.35665321e+00,  1.55627370e+00,  1.43848062e+00,  4.37909365e-01,
 2.20944834e+00,  3.91401529e+00,  3.83075571e+00,  3.74821711e+00,
 3.70781398e+00,  4.06578350e+00,  3.98652053e+00,  6.52369881e+00,
 6.55114841e+00,  6.46670485e+00,  6.49640560e+00,  6.38888311e+00,
 0.00000000e+00]), array([ 7.49168033e-03, -4.68174368e-03, -1.10390484e-02, -1.10584497e-02,
 1.41388178e-03, -1.06775761e-02,  1.00581646e-02,  3.75399590e-02,
 8.41850281e+00,  8.44219494e+00,  8.45814991e+00,  8.38665771e+00,
 8.40281391e+00,  8.50259399e+00,  8.57677078e+00,  3.58961105e-01,
 3.64523947e-01,  4.08766508e-01,  9.39686894e-01, -1.10283673e+00,
-1.08319187e+00, -1.09139729e+00, -1.11675060e+00, -1.15228677e+00,
-1.09774292e+00, -1.19562531e+00, -1.13858557e+00, -1.15359569e+00,
-1.12939239e+00, -1.12373853e+00, -1.11977851e+00, -1.19691634e+00,
-1.28741932e+00, -1.26653290e+00,  3.39888811e-01,  4.09049463e+00,
 3.80493259e+00,  4.03406811e+00,  4.13261795e+00,  4.17006302e+00,
 3.98356605e+00,  4.00270844e+00,  4.19106960e+00,  4.18228388e+00,
 4.15771580e+00,  1.71551049e+00,  1.74879837e+00,  1.74970222e+00,
 1.55832195e+00,  1.46623445e+00,  1.52128768e+00,  1.47478819e+00,
 1.45696664e+00,  1.51907384e+00,  1.59289932e+00,  1.72701406e+00,
 1.68308496e+00,  1.41604972e+00,  1.37924576e+00,  1.34472179e+00,
 1.42284131e+00,  1.44056916e+00,  1.42887235e+00,  1.00644696e+00,
 1.33061457e+00,  1.52866602e+00,  1.48268652e+00,  2.93390632e-01,
 2.37608671e+00,  3.86067653e+00,  3.85152316e+00,  3.84626269e+00,
 3.94109130e+00,  4.12743759e+00,  4.19520855e+00,  6.67722416e+00,
 6.68877792e+00,  6.53988838e+00,  6.55378771e+00,  6.53235960e+00,
 0.00000000e+00]), array([ 7.15290289e-03, -3.76381376e-03, -4.35325503e-03,  3.40498686e-02,
-5.77181578e-03, -8.01658630e-03,  3.92428041e-03, -6.40487671e-03,
 4.33539915e+00,  4.38108110e+00,  4.38908911e+00,  4.31167459e+00,
 4.44889736e+00,  4.43891478e+00,  4.44444370e+00, -5.81238270e-02,
-6.33699298e-02, -2.15378404e-02,  3.27619314e-02, -1.98275352e+00,
-1.98835027e+00, -1.97333574e+00, -2.01061344e+00, -1.75364959e+00,
-1.73991680e+00, -1.74862218e+00, -1.68127429e+00, -1.67997742e+00,
-1.62858677e+00, -1.60475683e+00, -1.67112112e+00, -1.59123802e+00,
-1.81741285e+00, -1.74995410e+00,  5.45792341e-01,  3.53152752e+00,
 3.44049740e+00,  3.48952127e+00,  3.51746535e+00,  3.56565976e+00,
 3.55737424e+00,  3.58430362e+00,  3.74230790e+00,  3.71496391e+00,
 3.71704817e+00,  1.32301080e+00,  1.32397556e+00,  1.36878800e+00,
 1.11489272e+00,  1.08664441e+00,  1.05647612e+00,  1.06448281e+00,
 1.13572860e+00,  1.03527439e+00,  9.46013331e-01,  9.80177045e-01,
 1.01796496e+00,  6.63047910e-01,  7.25941300e-01,  8.46044838e-01,
 9.60502267e-01,  9.15119648e-01,  7.72390723e-01,  3.22260946e-01,
 6.89065456e-01,  9.13342834e-01,  8.54033232e-01, -3.93604040e-02,
-1.43246174e-01,  1.73273277e+00,  1.50699842e+00,  1.42235255e+00,
 1.56513536e+00,  1.79288483e+00,  1.75213313e+00,  4.17738152e+00,
 4.21064568e+00,  4.25692749e+00,  4.20731449e+00,  4.26317978e+00,
 0.00000000e+00]), array([-3.56145436e-03, -2.45989580e-03, -1.36526227e-02,  4.83196974e-03,
-1.26029253e-02,  1.04360402e-01, -1.49490178e-01, -9.83047485e-03,
 1.01106482e+01,  1.01594315e+01,  1.01241169e+01,  1.01030016e+01,
 1.02209444e+01,  1.02338295e+01,  1.02098627e+01,  5.54937124e-01,
 5.56956053e-01,  5.30281663e-01,  5.79059958e-01,  1.76383388e+00,
 1.76415682e+00,  1.78038621e+00,  1.69284463e+00,  1.93894184e+00,
 2.05576372e+00,  1.94724560e+00,  2.06363964e+00,  2.11163998e+00,
 2.21606398e+00,  2.23521996e+00,  2.21502066e+00,  2.18667722e+00,
-4.05831051e+00, -3.94848633e+00, -1.53277540e+00,  1.32030487e+00,
 1.21679044e+00,  1.15970564e+00,  1.20855880e+00,  1.28216004e+00,
 1.26076221e+00,  1.33275700e+00,  1.56388092e+00,  1.47118139e+00,
 1.50467539e+00,  2.12888670e+00,  2.04766560e+00,  2.08892894e+00,
 1.91710329e+00,  1.81666064e+00,  1.82030821e+00,  1.79003668e+00,
 1.80875301e+00,  1.78289342e+00,  1.72387981e+00,  1.84603930e+00,
 1.84912777e+00,  1.57782173e+00,  1.52175188e+00,  1.60484016e+00,
 1.65300369e+00,  1.67188549e+00,  1.53343439e+00,  1.03925228e+00,
 1.43717790e+00,  1.65788364e+00,  1.63817692e+00,  7.39932060e-01,
 3.30561090e+00,  2.13822937e+00,  1.91578436e+00,  1.70660138e+00,
 7.87102795e+00,  8.22260857e+00,  8.11494350e+00,  7.52737951e+00,
 7.59716177e+00,  7.57740450e+00,  7.63630867e+00,  7.65558624e+00,
 0.00000000e+00]), array([-2.43612053e-03, -4.76111425e-04, -9.71215963e-03,  1.50212646e-03,
-1.04947686e-02,  4.44687679e-02,  2.04714425e-02,  1.01037979e-01,
 7.72188997e+00,  7.74100924e+00,  7.79031515e+00,  7.74492788e+00,
 7.85218239e+00,  7.84209061e+00,  7.85480499e+00,  4.56170261e-01,
 4.24246907e-01,  3.61471891e-01,  4.71042752e-01, -1.37977719e+00,
-1.38724470e+00, -1.37706351e+00, -1.39536107e+00, -1.39366853e+00,
-1.28558826e+00, -1.36817527e+00, -1.29322362e+00, -1.29349303e+00,
-1.30371404e+00, -1.19259501e+00, -1.22906113e+00, -1.23999190e+00,
-1.23326492e+00, -1.15068233e+00,  1.29955387e+00,  4.19632196e+00,
 3.93904829e+00,  4.00363159e+00,  4.05381727e+00,  4.02277851e+00,
 3.94122195e+00,  3.72968793e+00,  3.98739171e+00,  3.98718596e+00,
 3.95330143e+00,  1.47667003e+00,  1.46422613e+00,  1.40193534e+00,
 1.31831741e+00,  1.38921595e+00,  1.46310365e+00,  1.45370543e+00,
 1.43666434e+00,  1.52535069e+00,  1.57404947e+00,  1.68225265e+00,
```

```
 1.56646371e+00,  1.37401080e+00,  1.37844229e+00,  1.41454148e+00,
 1.38707066e+00,  1.38469875e+00,  1.39444518e+00,  9.48256969e-01,
 1.23164034e+00,  1.45208657e+00,  1.39723492e+00,  1.09957144e-01,
 1.75547695e+00,  3.55722380e+00,  3.52614045e+00,  3.40922451e+00,
 3.34893179e+00,  3.75083971e+00,  4.02365303e+00,  6.45244217e+00,
 6.21236324e+00,  6.14101410e+00,  6.15322924e+00,  6.10937023e+00,
 0.00000000e+00]), array([ 1.89943202e-02, -8.47208034e-03, -5.04800677e-03,  1.75801367e-02,
-7.35218823e-03, -3.40402126e-03,  7.90405273e-03,  1.15947723e-02,
 4.63532352e+00,  4.68507576e+00,  4.68959236e+00,  4.65207386e+00,
 4.75351048e+00,  4.74721289e+00,  4.77329302e+00,  1.29953623e-02,
-3.10603380e-02,  1.70078278e-02,  3.67739201e-02, -1.97280836e+00,
-1.98672736e+00, -1.96096718e+00, -1.97405267e+00, -1.86769438e+00,
-1.83868837e+00, -1.87148023e+00, -1.80225360e+00, -1.75440300e+00,
-1.74171567e+00, -1.65932941e+00, -1.73444784e+00, -1.71204257e+00,
-1.68360710e+00, -1.60183096e+00,  7.48179674e-01,  3.65722227e+00,
 3.58675122e+00,  3.71549726e+00,  3.74162674e+00,  3.67946434e+00,
 3.68425298e+00,  3.62118840e+00,  3.82954764e+00,  3.83099222e+00,
 3.81114483e+00,  1.19418287e+00,  1.17958772e+00,  1.21745646e+00,
 9.73472595e-01,  9.90650773e-01,  1.03591788e+00,  1.05414927e+00,
 1.01532459e+00,  1.14738607e+00,  1.14464438e+00,  1.19743168e+00,
 1.12569010e+00,  9.49953079e-01,  9.01930749e-01,  8.92324924e-01,
 8.48129392e-01,  9.70325112e-01,  8.97294521e-01,  5.43425560e-01,
 7.89195716e-01,  1.02131879e+00,  1.01424551e+00,  4.13750410e-02,
 7.98442364e-02,  2.00400496e+00,  1.83129418e+00,  1.67243886e+00,
 1.57274151e+00,  1.81651533e+00,  1.85329843e+00,  4.52320147e+00,
 4.43795061e+00,  4.33450031e+00,  4.40710640e+00,  4.39023495e+00,
 0.00000000e+00]), array([ 1.61145460e-02, -5.21355122e-03, -5.88311255e-03,  3.13270837e-03,
-1.93266273e-02, -6.58571720e-04, -1.40205771e-02,  4.73184586e-02,
 5.21541309e+00,  5.26382256e+00,  5.26358461e+00,  5.23867130e+00,
 5.34129095e+00,  5.33169746e+00,  5.36014843e+00,  1.08132094e-01,
 5.00658751e-02,  8.98392200e-02,  1.23676538e-01, -1.80941367e+00,
-1.80852580e+00, -1.82363987e+00, -1.84208775e+00, -1.78697860e+00,
-1.75317419e+00, -1.80120623e+00, -1.71550274e+00, -1.65412533e+00,
-1.64322746e+00, -1.55023050e+00, -1.61376143e+00, -1.63452637e+00,
-1.60035706e+00, -1.57095027e+00,  8.40176344e-01,  3.68868637e+00,
 3.60085964e+00,  3.74211979e+00,  3.79608679e+00,  3.67407322e+00,
 3.72949862e+00,  3.67061090e+00,  3.89826441e+00,  3.90205359e+00,
 3.85769320e+00,  1.16259682e+00,  1.15809298e+00,  1.12973022e+00,
 9.04931307e-01,  1.00681674e+00,  9.86222506e-01,  1.02426755e+00,
 9.80387688e-01,  1.11042380e+00,  1.19531822e+00,  1.26922464e+00,
 1.10207796e+00,  1.01868665e+00,  1.06743383e+00,  1.05469036e+00,
 1.01646256e+00,  1.15460849e+00,  1.15468931e+00,  7.70035923e-01,
 1.00410342e+00,  1.27625549e+00,  1.25915587e+00,  4.73923683e-02,
 4.22525883e-01,  2.28506947e+00,  2.18154407e+00,  1.99787259e+00,
 1.92780006e+00,  2.20947790e+00,  2.19269633e+00,  4.99236107e+00,
 4.87909269e+00,  4.74363661e+00,  4.62645912e+00,  4.56189728e+00,
 0.00000000e+00]), array([ 1.71460211e-02, -6.78758509e-03, -6.64265454e-03, -3.64554999e-03,
-2.04624534e-02,  3.17335129e-04, -1.09669566e-03,  3.54952812e-02,
 5.72058487e+00,  5.74009848e+00,  5.74295425e+00,  5.73705864e+00,
 5.85630703e+00,  5.81816959e+00,  5.84033775e+00,  2.11078823e-01,
 1.57732189e-01,  1.84516430e-01,  2.36823320e-01, -1.63370228e+00,
-1.59904718e+00, -1.64595938e+00, -1.68681550e+00, -1.67288673e+00,
-1.62606907e+00, -1.66571403e+00, -1.57092142e+00, -1.58302689e+00,
-1.62252176e+00, -1.47066164e+00, -1.58123064e+00, -1.60473490e+00,
-1.61807632e+00, -1.56394219e+00,  8.57558727e-01,  3.70484567e+00,
 3.68277931e+00,  3.73811245e+00,  3.84082985e+00,  3.62049818e+00,
 3.74495745e+00,  3.72101283e+00,  3.96184301e+00,  3.91118002e+00,
 3.86400175e+00,  1.06792974e+00,  1.11559594e+00,  1.06831789e+00,
 8.63234103e-01,  9.90200281e-01,  9.74059939e-01,  9.83847499e-01,
 9.82832670e-01,  1.10246038e+00,  1.22444892e+00,  1.36928022e+00,
 1.08401322e+00,  1.09812009e+00,  1.19703102e+00,  1.23334837e+00,
 1.16844010e+00,  1.31220615e+00,  1.34278738e+00,  8.94194603e-01,
 1.20965767e+00,  1.52281260e+00,  1.47116482e+00, -1.10190153e-01,
 6.81061983e-01,  2.44612241e+00,  2.41813993e+00,  2.30916166e+00,
 2.26542783e+00,  2.56813216e+00,  2.49455905e+00,  5.37640190e+00,
 5.26374483e+00,  5.06210232e+00,  4.84744072e+00,  4.72886276e+00,
 0.00000000e+00])]
```

Расчет финальных показателей, только энергопотребление, только 20 первых зданий

Ввод [12]:
```
def calculate_model (x):
    lr = -1
    model = energy_lr[x.building_id][x.hour]
    if len(model) > 0:
        lr = np.sum([x[col] * model[i] for i,col in enumerate(lr_columns[3:])])
        lr += model[len(lr_columns)-3]
        lr = np.exp(lr)
    if lr < 0 or lr != lr or lr*lr == lr:
        lr = 0
```

```
    x["meter_reading"] = lr
    if x["row_id"] % 1000000 == 0:
        print ("Готово", x["row_id"])
    return x

results = results.apply(calculate_model, axis=1, result_type="expand")
```

Усечение данных до требуемого формата: row_id, meter_reading

Ввод [13]:
```
results_ready = pd.DataFrame(results, columns=["row_id", "meter_reading"])
```

Загрузка всех данных для заполнения их нулями

Ввод [15]:
```
results = pd.read_csv("http://video.ittensive.com/machine-learning/ashrae/test.csv.gz",
                      usecols=["row_id"])
results = pd.merge(left=results, right=results_ready, how="left",
                   left_on="row_id", right_on="row_id")
results.fillna(value=0, inplace=True)
print (results.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41697600 entries, 0 to 41697599
Data columns (total 2 columns):
row_id           int64
meter_reading    float64
dtypes: float64(1), int64(1)
memory usage: 954.4 MB
None
```

Выгрузка результатов в CSV файл
Итоговый файл занимает около 1 Гб

Ввод [16]:
```
results.to_csv("submission.csv",index=False)
```

Освобождение памяти

Ввод [17]:
```
del energy
del results
del results_ready
```

Часть 4
МОДЕЛИ КЛАССИФИКАЦИИ И ЕЁ МЕТРИКИ

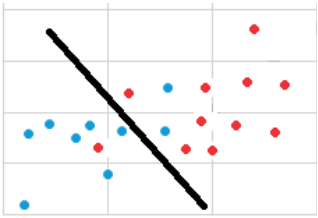
ТОЧНОСТЬ И ПОЛНОТА

Начиная разговор про задачи классификации задач машинного обучения и, в частности, обучения с учителем. Нам необходимо ввести некоторые метрики оценки качества работы модели для того, чтобы мы смогли оценить качество работы в задачах классификации. Почему нам нужно оценивать качество работы модели, потому что модели мы можем менять, выбирать, оптимизировать их гиперпараметры можем объединять модели в ансамбль и каждый раз нужно понимать лучше стало или хуже.

Так вот оценка качества работы моделей или ансамбля моделей позволяет нам сделать однозначный вывод на наших имеющихся данных лучше стало или хуже. Метрики расстояния или метрики оценки регрессионных задач обучения с учителем мало подходят для оценки качества работы модели в задачах классификации. Из-за того, что в задаче классификации обычно класс не числовой, он представляет некоторые название, то есть они вообще никак не связаны, либо некоторые ранг и расстояния между рангами. Что оценить качество работы модели требуется ввести некоторые дополнительные метрики.

Начнем с рассмотрения метрик задач бинарной классификации, предположим, что в почтовый ящик поступают письма и часть этих писем, нам нужна, а часть из них являются спамом. Рассмотрим на примере двадцати писем как будет работать бинарная классификация и как мы можем оценить качество работы моделей классификации по некоторым метрикам.

Предположим, что в некоторых координатах размер письма и срок регистрации домена, с которого пришло письмо, наши письма выглядит следующим образом синие точки не спам, а красные точки спам. У нас есть девять нормальных писем и одиннадцать писем из группы спам. Также у нас есть некоторая прямая которой мы разделили. Теперь нужно понять, насколько лучше мы можем предсказать данные, то есть улучшить работу нашей модели. Для этого вводим оценку.



Получается, что два нормальных письмо попали в спам, и одно спам письмо прошло в нормальные. Если сейчас оценить качество работы этой модели, например по безошибочности, то есть сколько правильно определенно категорий писем. Выходит, что правильно определено семь категорий для не спама и десять категорий для спама. Следовательно оценка безошибочности работы модели равна: 17/20 = 0,85.

В целом неплохо почти 0,9, можно сказать, что модель работает хорошо. Однако если мы, например, одно нормальное письмо перенесём из спама и единственное спам сообщение в нормальных перенесем в спам, путем изгибания линии. Получается, что модель научилась определять спам, но останется всё ещё одно нормальное письмо в спаме. Оценка безошибочности работы модели вырастит до 19/20 = 0,95. Однако мы будем терять одно письмо, а именно там могут быть важные сведения. Поэтому оценка безошибочности может применятся в базовых вещах, но, однако, нам примерно ничего не говорит о том насколько хорошо у нас модель стала работать. Так как мы можем продолжать терять нужную нам информацию. Хотя качество модели классификации может улучшиться.

Для того чтобы двигаться дальше с оценкой мы введем матрицу неточностей, она показывает какие у нас ошибки в целом могут быть. Для бинарной классификации выглядит не сложно.

Вверху записывают истинное значение, в нашем случае спам (цифра 1) или не спам (цифра 0). А в виде строк предсказанные значения.

		Истин.	
		0	1
Пред.	0	TP	FP
	1	FN	TN

Если мы сказали, что это не спам и это действительно не спам это называется TP – True Positive, то есть мы верно опередили тип письма. Если мы сказали, что письмо не спам, а это письмо спам это называется FP – False Positive, то есть мы ошиблись с положительной оценкой. Если письмо попало в спам, а оно является нормальным это FN – False Negative, то есть ошиблись с негативным предсказанием. И остается последний вариант, когда письмо со спамом попадает в спам это TN – True Negative, то есть мы верно определили спам.

Подставим наши значения в матрицу:

		Истин.	
		0	1
Пред.	0	7	1
	1	2	10

Чтобы вычислить точность нужно:

$$\text{Точность} = \frac{TP}{TP + FP}.$$

Точность показывает отношения правильных положительных предсказаний, к всем положительным предсказаниям. В нашем случае точность = 7/8.

Есть еще одна метрика полнота:

$$\text{Полнота} = \frac{TP}{TP + FN}.$$

Полнота показывает сколько точно определенно положительных экземпляров из всех действительно положительных объектов выборки. В нашем случае полнота = 7/9.

Можно задать вопрос, почему не используются другая строка или столбец. Однако использование ещё двух пересечений будет избыточно. Из-за того, что, если взять суммы по столбцам, то получится 11 и 9, а суммы по строкам будут 8 и 12. Соответственно общие количество столбцов и строк — это число всех элементов, а по строкам по столбцам у нас реальное либо предсказанное количество элементов.

Поэтому достаточно двух метрик и общего количества объектов для того, чтобы полностью описать поведение нашей системы и дополнительно вводить ещё две метрики нет нужды, потому что они будут следовать сложением, вычитанием, умножением делением всех остальных.

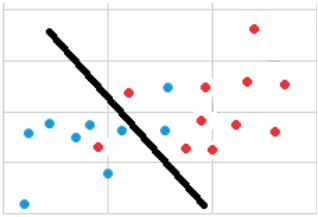
И уже ориентируясь на метрики точности и полноты. Мы можем улучшить нашу оценку качества работы модель, даже сейчас видно, что, если будем ориентироваться на полноту, то есть на количество пропущенных в спам писем это уже даст нам достаточно хорошую метрику оценку качества работы модели. Поскольку мы не хотим, чтобы в спам попали важные письма и поэтому полнота должна быть равна единице. И как только получим полноту равную единице, далее уже будем оценивать по точности работы, то есть сколько спама мы не пропустили. Если полнота не равна единице, то мы будем работать только на увеличение полноты. Такой подход, когда точность и полноту мы можем оценивать независимо, даёт гораздо больше свободы в оценке качества работы модели.

F-MEPA

Продолжая про меры оценки моделей в задачах классификации. Рассмотрим, как полнота и точность могут быть сведены к одной оценке и как мы можем получить некоторую универсальную оценку для нашей задачи по определению писем спам это или не спам.

Синие точки хорошие письма, красные точки – спам. И мы определять эти письма лучше и добиться максимальной точности. Для этого мы подсчитали точность, то есть оценку *Precision*. Также вычисли полноту, то есть оценку *Recall*. Теперь можно вычислять *F-меру*:

$$F = \frac{2 * Presision * Recall}{Presision + Recall}.$$



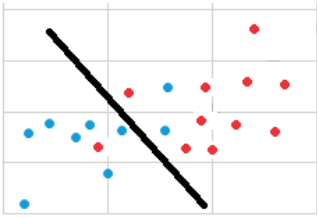
Если подставить в формулу наши значения, то получим, что $F = 0{,}82$ это и есть мера для нашей модельной задачи. Она достаточно близко находится без ошибочности. Однако при улучшении как полноты так точности она будет меняться и нам хотелось бы каким-то образом управлять, поскольку здесь средняя гармоническая соответственно мы в равной степени учитываем как точность, так и полноту. Но для нашей задачи важнее учитывать полноту чем точность тогда мы могли снизить влияние точности в этой формуле.

Для этого вводят параметризированную *F-меру*:

$$F_\beta = \left(1+\beta^2\right)\frac{Presision * Recall}{\beta^2 * Presision + Recall}.$$

Получается некоторая взвешенная оценка *F-меры*. Поскольку за счёт β параметра, который может принимать значение от нуля до единицы мы можем управлять насколько мы хотим учитывать точность, а насколько полноту. Если взять $\beta = 0.1$, то точность почти не будет влиять на F_β-*меру*. Это позволит более качественно решить нашу задачу по отсечению спама.

F-мера и F_β-*мера* являются базовыми оценками и для бинарной классификации, в основном используются именно они. Также мы можем изогнуть и сторону в точности, то есть нам нужно, чтобы точно определили все нужные классы.



Например, синие точки – это благонадёжные заёмщики в банке, а красные точки неблагонадежные. Банку главное, чтобы кредит был возвращен. Поэтому нам нужны только благонадежность заёмщики. Из-за этого точность будет намного важнее чем полнота и соответственно β будет около единицы. Поскольку нас не так волнует сколько потеряем благонадежных заемщиков, главное, чтобы все заемщики, которых мы выявили как благонадежных ими же и являлись.

Таким образом управляя параметром *β,* мы можем изогнуть оценку *F-меры*, в нужную нам сторону.

ROC AUC И GINI

В случае, когда затруднительно использовать *F-меру* или F_β*-меру* примеру в задача бинарной классификации для оценки качества работы моделей или нужно получить дополнительную оценку для того, чтобы её соотнести с *F-мерой*, то есть нужна ещё одна независимая оценка качества работы модели, либо какая-то графическая интерпретация качества работы модели, то используют *ROC AUC*. *ROC* это кривая качества работы классификатора, а *AUC* это площадь под этой кривой. Рассмотрим эту кривую, она строится в координатах *TPR (TruePositiveRate)* и *FPR(FalsePositiveRate)*.

Продолжаем работать с нашей матрицей неточности бинарного классификатора.

TP=7	FP=1
FN=2	TN=10

Вычислим *TPR*:

$$TPR = \frac{TP}{TP + FP}.$$

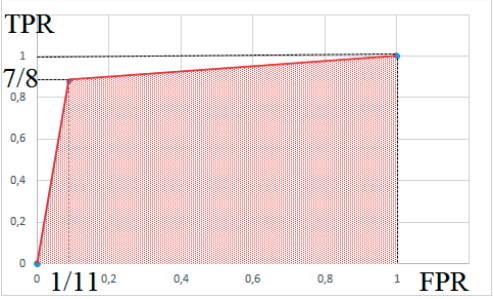
Для нашего примера *TPR = 7/8.* Вычислим *FPR:*

$$FPR = \frac{FP}{TN + FP}.$$

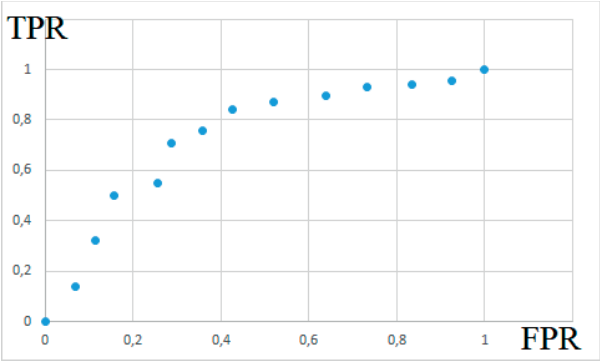
В нашем случае *FPR* = 1/11. Еще один критерий – это специфичность модели:

$$\text{Специфичность} = 1 - FPR.$$

Данный критерий показывает, насколько модель не пропускает ошибки. В нашем случае он равен 10/11. То есть модель не пропускает ошибки в случаях 10/11. Построим *ROC* кривую:

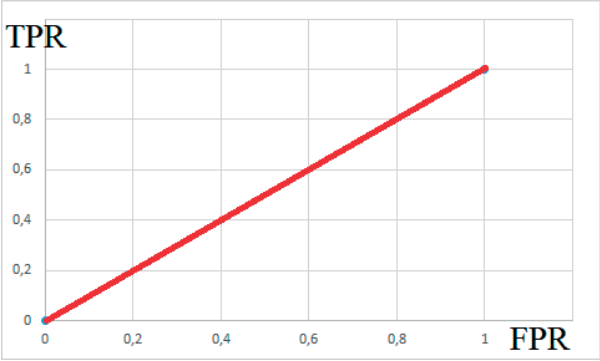


Поскольку у нас один единственный случай, то точка на кривой будет одна с координатами (*1/11;7/8*). Соответственно закрашенная часть это *ROC AUC*, то есть площадь по *ROC*. В случае бинарной классификации *ROC* кривая выглядит просто, как соотношение *TPR* и *FPR*. В нашем случае *ROC AUC* равна *TPR,* то есть *7/8*. Если у нас есть ряд гиперпараметров и нужно оценить одну или несколько моделей по их гиперпараметрам. То в этом случае у нас будет несколько точек на графике, и они будут выглядеть следующим образом:

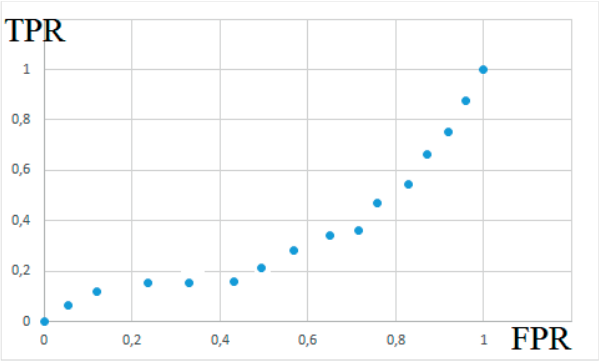


Эти точки также выходят из нуля и заканчивается единицей. Соответственно набор этих точек каким-то образом определенный, каждая точка соответствует значению гиперпараметра или набора гиперпараметров модели, которые определяют её специфику работы, точность. Набор этих точек сложится в *ROC* кривую и уже по площади под этой *ROC* кривой мы сможем использовать эту площадь как дополнительный критерий.

Понятно, что у нас, может быть, случай, например когда кривая начинается в нуле и заканчивается единицей и она представляет собой прямую линию. Однако на практике такое может случится только если случайным образом раскидывать классы.



Также в ряде случае *ROC* кривая пройдет ниже, чем случайный выбор. То есть будет хуже, чем случайны выбор. Но это не плохо, потому что мы можем в случае бинарного классификатора легко обратить предсказания классификатора, то есть ноль сделать единицей, а единицу нулем. Тогда кривая отразится относительно диагонали, и мы получим удовлетворяющую нашим критериям кривую.



Данные критерии отлично подходят для того, чтобы нам графически понять, как ведут себя разные модели или при разных наборах гиперпараметров, ведёт себя одна и та же модель.

Также дополнительно поскольку *ROC AUC* не меньше 1/2, то вводят некоторый критерий *Gini*, который равен:

$$Gini = ROC\ AUC * 2 - 1.$$

Критерий *Gini* является нормировочным для *ROC AUC*. Поскольку принимает значения от нуля до единицы.

Подытожим *ROC* кривая показывает качество работы классификатора в случае бинарной классификации, строится в координатах *TPR* и *FPR*. Построив *ROC* кривую, мы смотрим на площадь под ней, то есть по факту берём интеграл по значению правильных ответов на назначение ошибок. И зависимости от интегрирования мы получаем конечный критерий.

Также особой важностью форма *ROC* кривой, поскольку можно ориентироваться на то, чтобы она как можно раньше доходила до единицы. Благодаря такой форме кривая может быть предпочтительней чем кривая с большей площадью. Например, банковская сфера, единица будет показывать процент благонадежных заемщиков. И поскольку важно увеличить именно этот процент, а не в целом точность предсказаний, то такая кривая будет лучше подходить под эти условия.

ОЦЕНКА КАППА КОЭНА

Когда мы говорим про задачи бинарной классификации, то есть у нас всего два класса: верно, неверно, спам не спам, должник кредитор, холодно жарко и так далее. То *F-мера* или F_β*-мера* являются достаточно они являются достаточно хорошими оценками качества работы модели машинного обучения.

Однако если классов больше, чем два, то есть задача уже множественной классификации, матрица неточности будет содержать больше строк и столбцов и непонятно что из этого подставлять в *F оценку* и как она должна выглядеть, если значений в нашей матрице больше четырех. Для этих случаев используют другие оценки, одна из этих *Каппа-оценка Коэна*. Она может работать с матрицей неточности любого размера. Она обобщает подход к оценке качества работы.

Рассмотрим, как её можно вычислить. Будем искать для двух классов чтобы просить упростить вычисления, однако она может быть вычислена для любого количества классов. Вводится сумма строк и столбцов. То есть:

$a_{11} = 7$	$a_{21} = 1$	$m_1 = 8$
$a_{12} = 2$	$a_{22} = 10$	$m_2 = 12$
$n_1 = 9$	$n_2 = 11$	

Теперь вычисляется P_0:

$$P_0 = \frac{\sum_{i=1}^{k} a_{ii}}{N} = \frac{(7+10)}{20},$$

где k – это количеству классов, а N сумма элементов по диагонали. В нашей ситуации получаем 0,85. Сложнее с оценкой реальной ситуации. *Каппа-оценка Коэна* базируется на предсказанной ситуации и на реальной ситуации, то есть фактической вероятности тех элементов что у нас есть. Для этого для этого вычисляется вероятность предсказанная и вероятность фактическая.

Вводится параметр P_e:

$$P_e = \frac{\sum_{i=1}^{k} n_i * m_i}{N^2} = \frac{(9*8+11*12)}{20^2} = 0,51.$$

Теперь можем вычислить *Каппа-оценка Коэна*:

$$\kappa = \frac{P_0 - P_e}{1 - P_e}.$$

Каппа оценка должна быть больше, чем *фактическая вероятность* P_e. Если *Каппа оценка* меньше нуля, то модель полностью не работает поскольку

случайный выбор даст более лучший результат. В нашем случае *Kappa оценка* равна примерно 0,69. Говорят, что если *Kappa* от 0,5 до 0,75, то это достаточно хороший классификатор, больше 0,75 это уже отличный классификатор.

В итоге *Kappa оценка* позволяет нам расширить случай бинарной классификации на множественные классификации. Поскольку формулы позволяют использовать их с каким угодно количеством классов.

ВЗВЕШЕННАЯ КВАДРАТИЧНАЯ ОЦЕНКА КАППА КОЭНА

Квадратично взвешенный коэффициент *Kappa Коэна* вводится в тех случаях, когда есть дополнительная информация о задаче классификации, а именно что у нас ранговая классификация. То есть в исходных данных они каким-то образом упорядочены и есть понимание что на сколько-то классов они расположены по порядку возрастания, и мы хотим каким-то образом уменьшить штраф если у нас небольшой промах между классами и увеличить штраф есть у нас большой промах между классами.

Для этого вводится матрица весов для того, чтобы мы могли от ранжировать промахи между классами. Это возможно только когда у нас задача ранговой классификации. То есть у нас числа, которым ставим соотношения наши объекты. Если у нас категории, то взвешенный коэффициент *Kappa Коэна* скорее всего не применим поскольку категории друг с другом не связаны. А в случае мы можем применить взвешенную оценку.

Она базируется на матрице весов, то есть на некоторых наших предположениях о том, как связаны ранговые классы и логичным было бы, например, сделать разницу между рангами равной единице. То есть в случае двух классов получаем следующую матрицу весов w_{ij}:

0	1/2
1/2	0

Если элемент находится по находится по диагонали это значит он точно попал в предсказания, то есть либо это *TruePositive*, либо *TrueNegative*. Другими словами, мы точно предсказали верное значение, либо точно предсказали неверное значение.

Поэтому в матрице весов элементы по диагонали всегда будут нули, нам не нужно как-то нормировать верные предсказания. Нам нужен нормировать ошибку. И, например, задать здесь 1/2, поскольку можем сказать, что у нас всего два класса и на один класс отодвинулись от правильного.

Однако *квадратичная Kappa Коэна* базируется на квадратах разности расстояний. Поэтому для двух классов матрица будет:

0	$(1/2)^2 = 1/4$
$(1/2)^2 = 1/4$	0

А в случае трех классов:

0	$(1/3)^2 = 1/9$	$(2/3)^2 = 4/9$
$(1/3)^2 = 1/9$	0	$(1/3)^2 = 1/9$
$(2/3)^2 = 4/9$	$(1/3)^2 = 1/9$	0

Ну и так далее можно подсчитать ее для любого количества классов. В случае если у нас будет четыре класса, то 1/16, 4/16, 9/16 это расширяемо для любого количества классов.

Кроме матрицы весов для *квадратичной Каппы Коэна*, есть также матрица неточности E_{ij}. Возьмем матрицу неточностей из задачи про спам:

7	1
2	10

Также еще вводят матрицу вероятностей O_{ij}. Она представляет собой произведение матриц состоящих из сумм строк на суммы столбцов. И для нашего случая равна:

$$\begin{pmatrix} 8 \\ 12 \end{pmatrix} * \begin{pmatrix} 9 & 11 \end{pmatrix} = \begin{matrix} 72 & 88 \\ 108 & 132 \end{matrix}.$$

Матрица вероятностей описывает некоторое идеальное состояние, которое должно было бы случиться если бы мы предсказали правильно. То есть мы используем матрицу неточности E_{ij} и матрицу вероятностей O_{ij} чтобы соотнести случившееся состояние классификатора с реальным состоянием.

После вычисления матриц их нужно нормировать. То есть нормировочный коэффициент в матрице неточности равен 1/20, а в матрице вероятностей $(1/20)^2 = 1/400$. После вычисления и нормирования трех матриц: весов, неточностей, вероятностей, нужно получить взвешенное значение неточности и взвешенное значение вероятностей. Взвешенное значение неточности равно:

$$e = \sum_{ij} W_{ij} * E_{ij} = \frac{1}{20} * \left(\frac{1}{4} * 1 + \frac{1}{4} * 2 \right).$$

Получается, что мы отбрасываем верные значения за счет нулей в матрице весов. 1/20 это нормировочный коэффициент. Взвешенность нужна только для того, чтобы мы могли каким-то образом от ранжировать ошибки близких классов от ошибок далёких классов. Также вычисляется взвешенное значение вероятностей:

$$o = \sum_{ij} W_{ij} * O_{ij} = \frac{1}{400} * \left(\frac{1}{4} * 88 + \frac{1}{4} * 108 \right).$$

Теперь мы должны понять, насколько e близко к нулю. То от нормировать на нашу ошибку и сравнить с нулём. Например, если мы ни разу не

ошиблись, то даже *e* взвешенное будет равна нулю. Рассчитываем *Kappa взвешенное*:

$$\kappa_w = 1 - \frac{e}{o}.$$

Для нашего случая она равна 0,69. И в нашем конкретном случае *взвешенная Kappa Коэна* в бинарной классификации, когда у нас два класса, дает ровно тот же результат что и не *взвешенная Kappa Коэна*. Поскольку в матрицу весов мы можем добавить любое число и от нормировать, то оно всё равно сократится и в итоге получится одно и тоже число. Поэтому *взвешенную Kappa Коэна* имеет смысл применять при количестве классов три или более.

Взвешенная Kappa Коэна совмещает в себе много подходов к оценке правильности работы классификатора. И особенно хорошо себя показывает, когда у нас большое число ранговых классов и нужно соотнести близкие и дальние промахи.

ЛОГИСТИЧЕСКАЯ ФУНКЦИЯ ПОТЕРЬ

В задачах классификации кроме непосредственной оценки качества работы модели также очень часто требуется функция ошибки, то есть некоторая функция, которая будет дифференцируемой, вдоль которой мы сможем двигаться чтобы эту ошибку минимизировать.

В задачах линейной регрессии у нас функция ошибки совпадала с оценкой это была сумма квадратов наших отклонений. Минимизировали сумму квадратов и двигались к минимальной ошибке и выходило точное решение. В задачах классификации не все методы позволяют получить точное решение. Некоторые методы двигаются посредством градиентного спуска, то есть двигается вдоль производной, вдоль направления уменьшения какой-то функции. И чтобы эту функцию определить, вводится функция *logloss*. Ее достаточно часто называют логистическая функция потерь иногда логарифмическая функция потерь. *logloss* выводится из функции правдоподобия, которая используется практически во всем машинном обучении. Эта функция правдоподобия должна всегда стремиться к максимуму. Вспомним как ее рассчитать для бинарной классификации:

$$L = \prod \left(a_i^{yi} * \left(1 - a_i\right)^{(1-y_i)} \right).$$

Получается, что $a_i^{y_i}$ вероятность получить один класс, и если мы этот класс не получили, то вероятность получить другой класс – это единица минус получить этот класс. Для того чтобы найти экстремум функции правдоподобия мы берем логарифм, потому что нужно от произведения перейти к сумме, тогда функция будет выглядеть как:

$$\ln L = \sum \ln(a_i)\, y_i + (1 - y_i) \ln(1 - a_i).$$

Поскольку у нас оба логарифма будут отрицательные, а функция должна стремиться к максимуму. Из-за того, что логарифм монотонная функция, то функция правдоподобия стремится к максимуму. Соответственно, чтобы был минимум ошибки, то есть минимум *logloss,* нужно повернуть в противоположную сторону:

$$logloss = -\sum y_i * \log a_i + (1 - y_i) \log(1 - a_i).$$

logloss всегда стремиться к минимуму, то есть в минимуме функции ошибки мы получаем максимум функции правдоподобия и реализуем нашу выборку. Получается, что принцип максимизации правдоподобия, который реализуется некоторыми функциями. Функция *logloss* отлично дифференцируется, она описывает все наши гиперпараметры и элементы выборки, но единственный минус из-за того, что она описывает только бинарную классификацию.

Что же делать если задача множественной классификации. Для них используется приём, называемый *сравнение один ко многим,* то есть для каждого класса должны вычислить его верные и неверные предсказания, посчитать по логической функции потерь. И далее считаем средние по всем классом. То есть в формулу нужно добавить коэффициент нормирования:

$$logloss = -\frac{1}{n} \sum y_i * \log a_i + (1 - y_i) \log(1 - a_i).$$

Рассмотрим пример есть следующая матрица неточности:

10	5	3	2
1	20	1	0
2	5	30	3
1	3	10	40

По диагонали расположены верные предсказания, остальные неверные. Соответственно *logloss* считаем по строкам в первой 10 верных предсказаний и (5 + 3 + 2) = 10. Во второй 20 верных и (1 + 1 + 0) = 2 неверных предсказаний. В третьей 30 верных и (2 + 5 + 3) = 10 неверных предсказаний. В четвертой 40 верных и (1 + 3 + 10) = 14 неверных предсказаний. То есть мы бинаризируем нашу мульти классовую классификацию и усреднением *logloss* по всем классам, и получаем некоторые направления куда нам нужно двигаться, какие гиперпараметры нужно поменять, чтобы уменьшить число ошибок и повысить точность работы нашей модели.

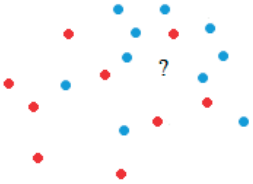
Если мы уменьшим число ошибок хотя бы каком в одном классе, то по любой метрике получим увеличение точности. Потому что любая метрика зависит от количества ошибок. А если количество ошибок перераспределится. То есть оно не изменится, то тогда *logloss* тоже не изменится.

МЕТОД БЛИЖАЙШИХ СОСЕДЕЙ

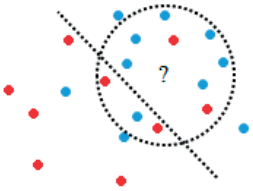
Метод ближайших соседей *(англ. kNN – k Nearest Neighbor)* при решении задач классификации машинного обучения является одновременно как одним из наиболее простых так одним из наиболее мощных. Некоторые авторы достаточно справедливо говорят, что правильное применение этого метода на очень больших выборках данных позволяет достичь наиболее точного решения, но есть проблему большой выборки данных, потому что метод начинает работать менее эффективно.

Сейчас рассмотрим подходы, которые позволяют это сделать, и часть из них разберём в следующих частях, как логическое продолжение этого метода, то есть метод решающих деревьев, случайного леса, бустинга, бэггинга и так далее. Всё они вытекают из метода ближайших соседей, который сам по себе является продолжением метода максимального правдоподобия.

В чём заключается метод ближайших соседей. Предполагаем, что у нас есть какой-то набор объектов:



Если работать по логистической регрессии, то есть просто разделения объектов на две группы, то будет достаточно много шума. Однако если мы возьмем неизвестный объект (знак вопроса), и мы хотим узнать к какому классу он принадлежит. Берём некоторую окрестность, она определяется числом *к*, отражающая количество соседей, которые в эту окрестность попадают. И по окружности считаем цвет точек, которые в нее попадают. Окружность зависит от числа точек, то есть нужно подсчитать одиннадцать объектов рядом.



По большинству по большинству этих объектов принимаем решение какой это класс. То есть *класс объекта = класс большинства соседей*. Если межклассовая граница лежит, то в районе границы достаточно уверенно определяем принадлежность класса, поскольку здесь больше синих объектов, то это синий объект. Метод ближайших соседей более толерантен к явной границе

разделения классов и позволяет эту границу нужным образом исказить, поскольку он оперирует локальным максимальным правдоподобием. Однако из преимуществ этого метода вытекают его проблемы.

Первая проблема – точность метода очень сильно зависит от числа от соседей. Если взять одного ближайшего соседа, то метод будет сильно шуметь, то есть брать все статистические ошибки и это будет недообучение. Если взять всю выборку как ближайших соседей, то будет переобучение.

Поэтому обычно выбирают некоторое значение как минимум в несколько раз меньше выборки или в районе максимального размера классов объектов. То есть у нас есть, например, 800 одних объектов выборки и 5 000 других объектов. Получается нужно выбрать не менее 800 чтобы достаточно достоверно определить класс объектов по методу ближайших соседей.

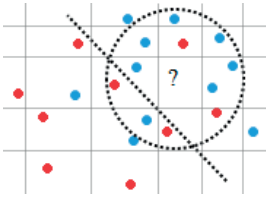
Следующей проблемой является проблема расстояния. Поскольку мы выбираем ближайших соседей нам нужно каким-то образом нормировать расстояние между точками. Из-за того, что при классификации могут быть категориальные переменные и причём они могут быть разноплановые, то есть могут быть категориальные переменные, могут быть номинативные числовые у них может быть разной диапазон.

Нужно это всё нормировать к одному диапазону, но даже после приведения к интервалу от 0 до 1 всех наших параметров, по которым принимаем решение. Возникает другая проблема «Проклятие большой размерности». Когда расстояние между удаленными объектами становиться примерно одинаковым. В случае разности по большому количеству параметров, то есть мы не можем достоверно, что объект нужного класса находится рядом или не рядом.

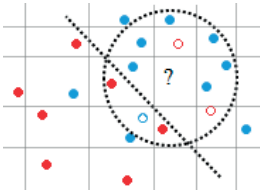
Приходится выделять главные признаки, выделить второстепенные и сначала посчитать расстояние по главным если по главным выходит двусмысленность, то считать уже по второстепенным параметрам расстояние и по ним уже принимать решение.

Третья проблема возникает поскольку метод ближайших соседей хорошо работает в том числе и на больших выборках, то нужно эти выборки и считать. Но например: у нас есть 100 ближайших соседей, получается 100 * 100 = 10000 различных расстояний, то есть от одного элемента до ста других. Выходит, 10.000 вычислений для того, чтобы классифицировать один элемент.

Для решения этой проблемы вводят сетку. Мы говорим, что параметры разбиваем на диапазоны и начинаем принимать решения в зависимости от этих диапазонов. Однако данный метод становится практически идентичен дереву принятия решений.



Еще одна проблема, которая возникает при использовании метода ближайших соседей. Нужно фильтровать исходные данные, то есть нам нужно, во-первых, определить, что у нас есть выбросы:



Пустые точки скорее всего являются выбросами, поскольку находятся далеко за границей классов. То есть возможно имело смысл сначала провести линию логистической регрессии и разделить классы, отбросить те элементы, которые слишком далеко от этих линий находятся, и затем по остальным принять решение.

Также проблема – это фильтрация малозначимых элементов. Поскольку количество синих элементов значительно больше красных, то мы могли бы их отбросить тем самым упростив вычисления. Из-за того, что плотность правильных элементов уже достаточна.

При грамотном решении этой проблемы метод ближайших соседей, дает очень хорошую точность. Хотя при большом количестве параметров и не очень большой выборке, получается разреженные пересечённые классы и не всегда понятно, где у нас выбросы и достаточная плотность.

В любом случае метод наименьших соседей является отличным базовым методом для того, чтобы получить какую-то первичную точность и в ряде задач позволяет достаточно хорошо приблизить уже финальное решение, но лучшая его реализация заключается в дереве принятия решений, которое рассмотрим позже.

ЧАСТЬ ПРАКТИЧЕСКИХ НАВЫКОВ К 4

СТРАХОВОЙ СКОРИНГ

Постановка задачи

Задача страхового скоринга: https://www.kaggle.com/c/prudential-life-insurance-assessment.

Требуется провести классификацию клиентов по уровню благонадежности для страхования жизни (всего 8 градаций) – Response. Для оценки доступно несколько параметров: виды страховки (Product_Info), возраст (Ins_Age), рост (Ht), вес (Wt), индекс массы тела (BMI), данные о работе (Employment_Info), данные страховки (InsuredInfo), история страхования (Insurance_History), семья (Family_Hist), медицинские данные (Medical_History) и медицинские термины (Medical_Keyword) – всего 126 переменных.

Загрузим данные и исследуем их. Найдем возможные «утечки» и взаимосвязи параметров для построения моделей.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [2]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import rcParams
import seaborn as sns
rcParams['figure.figsize'] = 16,8
```

Загрузка данных

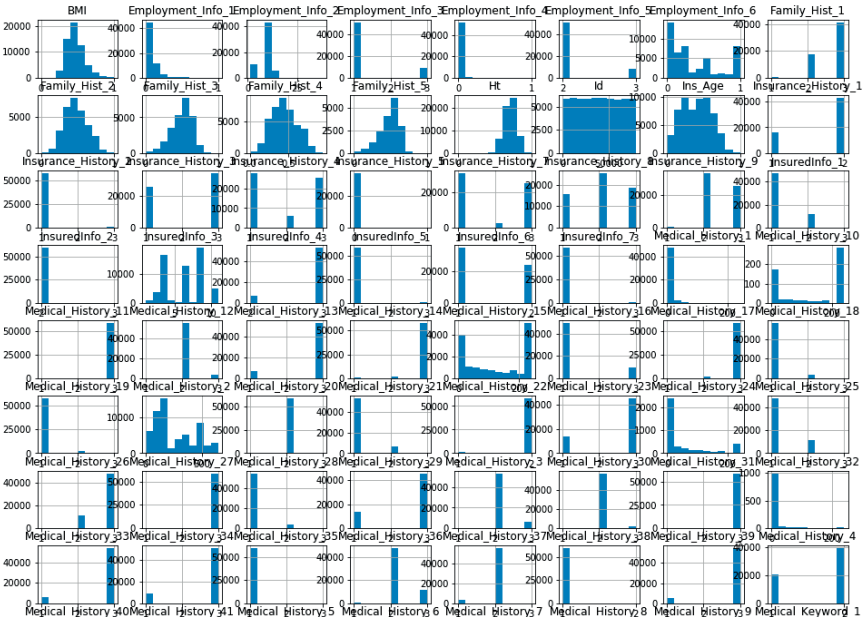
Ввод [3]:
```
train = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (train.info())
print (train.head())
```

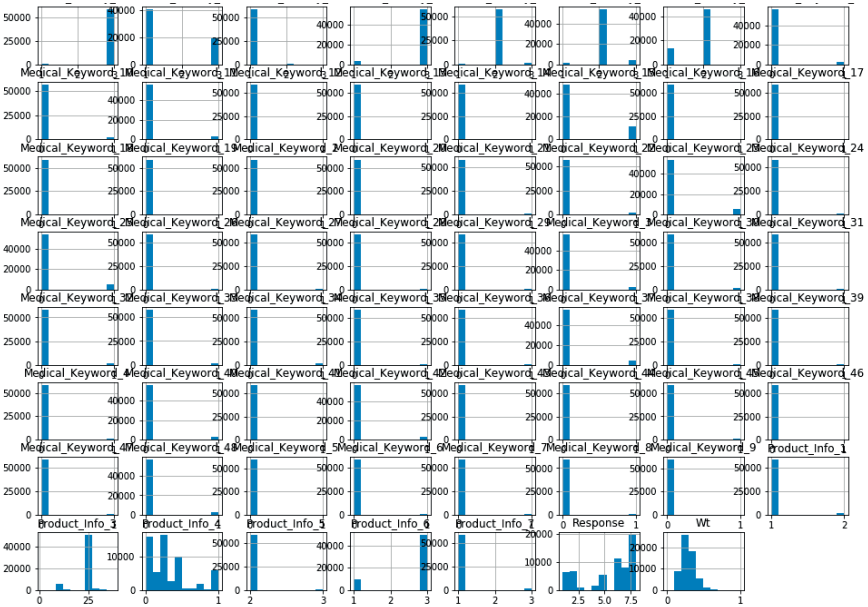
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
   Id  Product_Info_1 Product_Info_2  Product_Info_3  Product_Info_4  \
0   2               1             D3              10        0.076923
1   5               1             A1              26        0.076923
2   6               1             E1              26        0.076923
3   7               1             D4              10        0.487179
4   8               1             D2              26        0.230769
```

```
   Product_Info_5  Product_Info_6  Product_Info_7    Ins_Age        Ht  ...  \
0               2               1               1  0.641791  0.581818  ...
1               2               3               1  0.059701  0.600000  ...
2               2               3               1  0.029851  0.745455  ...
3               2               3               1  0.164179  0.672727  ...
4               2               3               1  0.417910  0.654545  ...

   Medical_Keyword_40  Medical_Keyword_41  Medical_Keyword_42  \
0                   0                   0                   0
1                   0                   0                   0
2                   0                   0                   0
3                   0                   0                   0
4                   0                   0                   0

   Medical_Keyword_43  Medical_Keyword_44  Medical_Keyword_45  \
0                   0                   0                   0
1                   0                   0                   0
2                   0                   0                   0
3                   0                   0                   0
4                   0                   0                   0

   Medical_Keyword_46  Medical_Keyword_47  Medical_Keyword_48  Response
0                   0                   0                   0         8
1                   0                   0                   0         4
2                   0                   0                   0         8
3                   0                   0                   0         8
4                   0                   0                   0         8

[5 rows x 128 columns]
```

Распределение значений

```
Ввод [4]: train.hist(figsize=(16,24), layout=(16,8))
          plt.show()
```

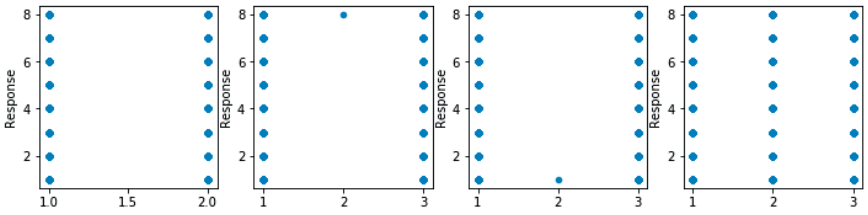


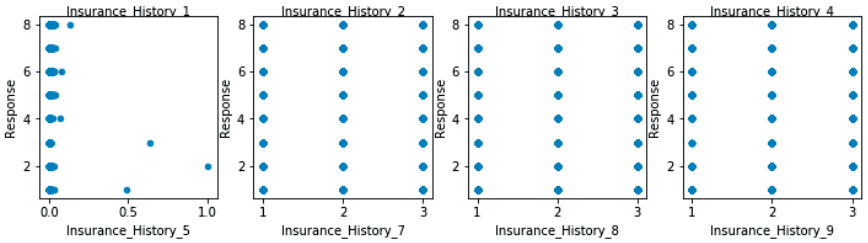


Зависимость скоринга от параметров: история страхования

Ввод [5]:
```
def data_correlation_plot (df, columns):
    rows = np.ceil(len(columns) / 4)
    fig = plt.figure(figsize=(12, rows*3))
    i = 1
    for column in columns:
        type_ = str(df[column].dtypes)
        if type_[0:3] == "int" or type_[0:5] == "float":
            area = fig.add_subplot(rows, 4, i)
            pd.DataFrame(df,
                columns=["Response", column]).plot.scatter(x=column,
                    y="Response", ax=area)
            i += 1
    plt.show()
```

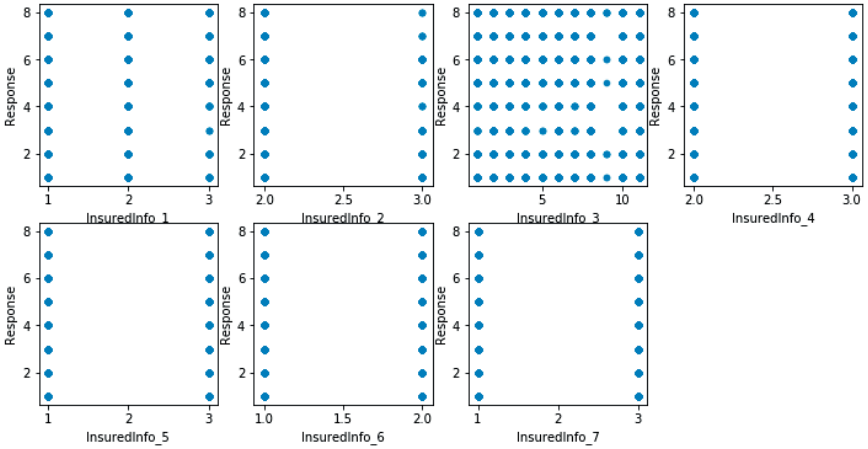
Ввод [6]:
```
data_correlation_plot(train,
    train.columns[train.columns.str.startswith("Insurance_History")])
```





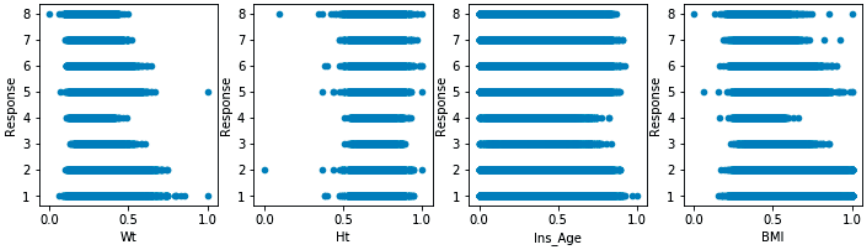
Зависимость скоринга от параметров: параметры страхования

```
Ввод [7]: data_correlation_plot(train,
              train.columns[train.columns.str.startswith("InsuredInfo")])
```



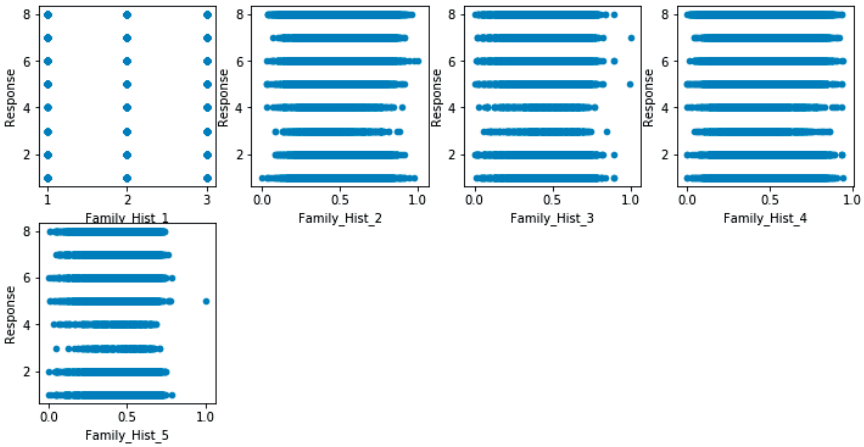
Зависимость скоринга от параметров: физиология

```
Ввод [8]: data_correlation_plot(train, ["Wt", "Ht", "Ins_Age", "BMI"])
```



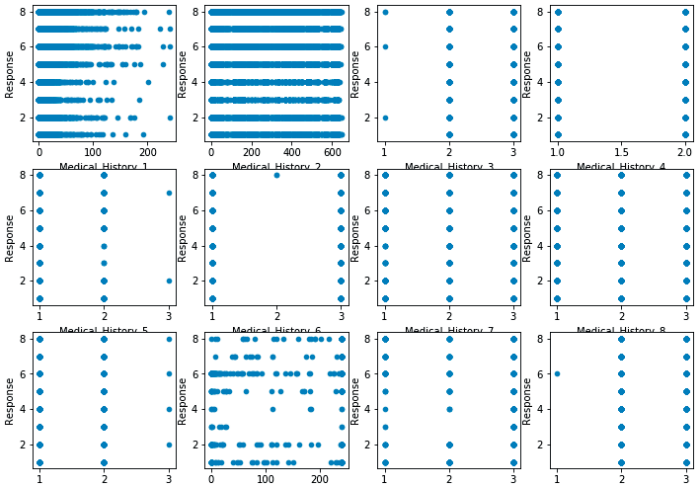
Зависимость скоринга от параметров: семья

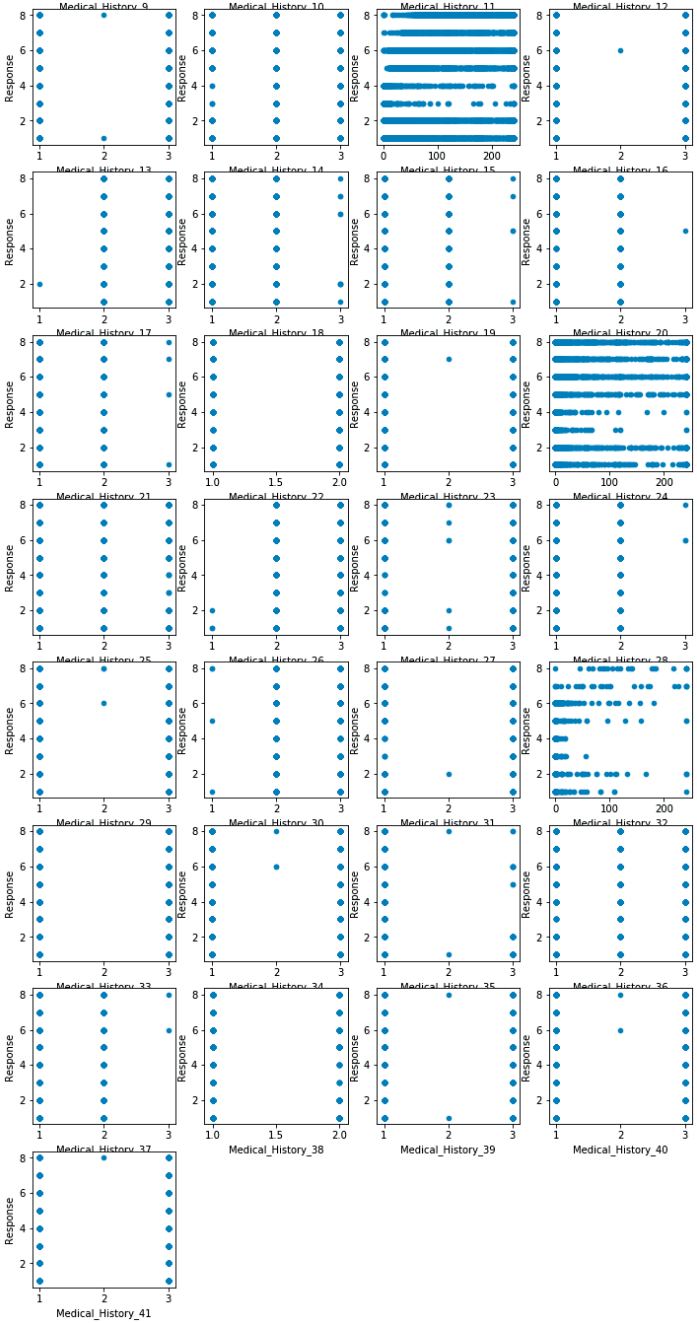
Ввод [9]:
```
data_correlation_plot(train,
    train.columns[train.columns.str.startswith("Family_Hist")])
```



Зависимость скоринга от параметров: здоровье

Ввод [10]:
```
data_correlation_plot(train,
    train.columns[train.columns.str.startswith("Medical_History")])
```

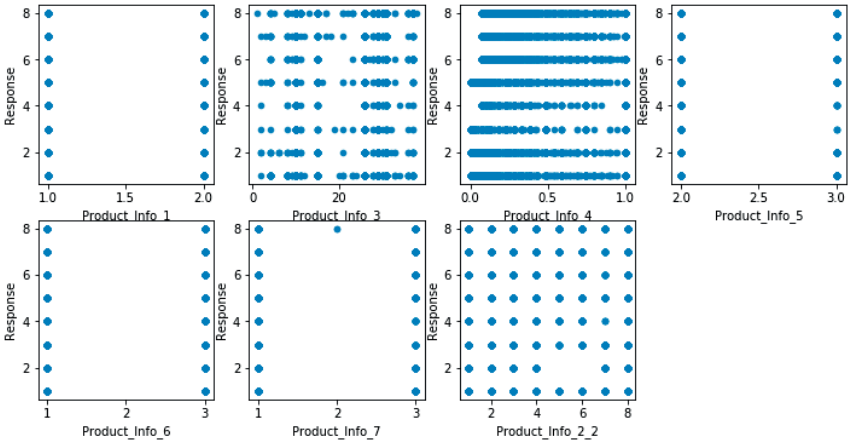



Medical_History_9
Medical_History_10
Medical_History_11
Medical_History_12
Medical_History_13
Medical_History_14
Medical_History_15
Medical_History_16
Medical_History_17
Medical_History_18
Medical_History_19
Medical_History_20
Medical_History_21
Medical_History_22
Medical_History_23
Medical_History_24
Medical_History_25
Medical_History_26
Medical_History_27
Medical_History_28
Medical_History_29
Medical_History_30
Medical_History_31
Medical_History_32
Medical_History_33
Medical_History_34
Medical_History_35
Medical_History_36
Medical_History_37
Medical_History_38
Medical_History_39
Medical_History_40
Medical_History_41
Response

Зависимость скоринга от параметров: страховка

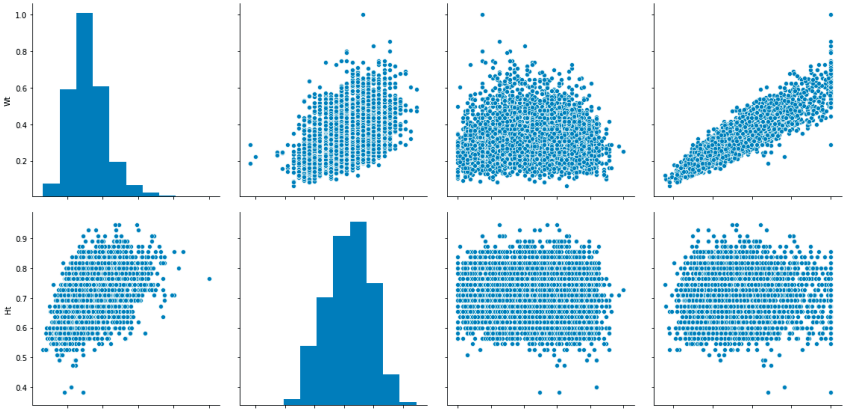
Ввод [11]:
```
train["Product_Info_2_1"] = train["Product_Info_2"].str.slice(0, 1)
train["Product_Info_2_2"] = train["Product_Info_2"].str.slice(1, 2).astype("int8")
train.drop(labels=["Product_Info_2"], axis=1, inplace=True)
```

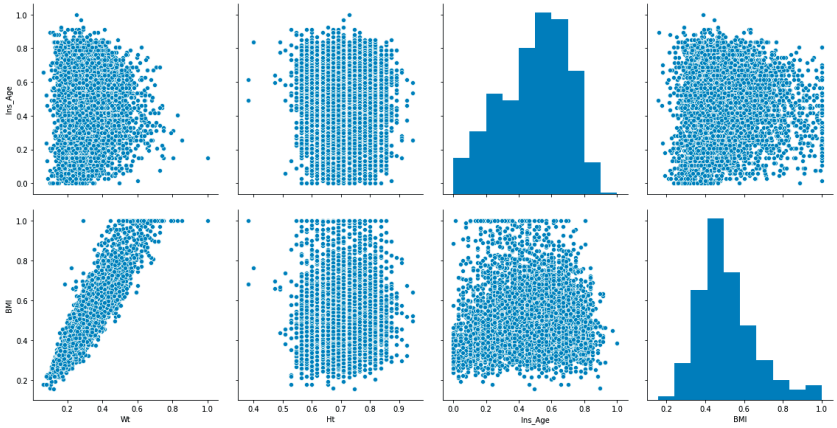
Ввод [12]:
```
data_correlation_plot(train,
    train.columns[train.columns.str.startswith("Product_Info")])
```



Взаимная корреляция биометрии

Ввод [14]:
```
data = pd.DataFrame(train[train["Response"]==1],
        columns=["Wt", "Ht", "Ins_Age", "BMI"])
sns.pairplot(data, height=4)
plt.show()
del data
```

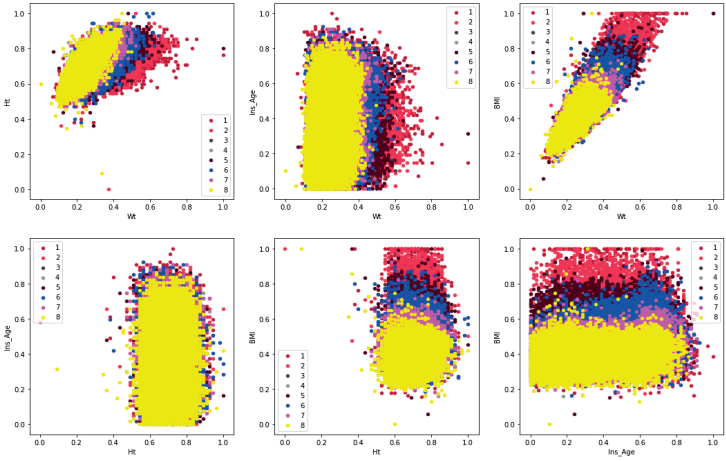




Кластеризация по биометрии

Ввод [17]:

```
columns_groups = [["Wt", "Ht"], ["Wt", "Ins_Age"], ["Wt", "BMI"],
                  ["Ht", "Ins_Age"], ["Ht", "BMI"], ["Ins_Age", "BMI"]]
colors = ["#A62639", "#DB324D", "#56494E", "#A29C9B", "#511C29",
          "#0000FF", "#FF00FF", "#FFFF00", "#00FFFF", "#00FF00"]
fig = plt.figure(figsize=(18,12))
i = 1
for c in columns_groups:
    data = pd.DataFrame(train, columns=c.append("Response"))
    legend = []
    area = fig.add_subplot(2, 3, i)
    for response in range(1, train["Response"].max() + 1):
        group = data[data["Response"] == response].plot.scatter(x=c[0],
            y=c[1], ax=area, c=colors[response-1])
        legend.append(response)
    area.legend(legend)
    i += 1
    del data
plt.show()
```



F1 И КАППА ОЦЕНКИ КЛАССИФИКАЦИИ

Постановка задачи

Загрузим данные и разделим выборку на обучающую/проверочную в соотношении 80/20.

Для обучающей выборки вычислим средние значения для веса, роста, индекса массы тела и возраста – для каждого из принятых решений. Предскажем оценку скоринга по близости данных средним значениям.

Проверим качество предсказания через F1-метрику и матрицу неточностей.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, confusion_matrix
from sklearn.metrics import cohen_kappa_score
```

Загрузка данных

Ввод [2]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Разделение данных на обучающие и проверочные, 80/20

Ввод [3]:
```
data_train, data_test = train_test_split(data, test_size=0.2)
print (data_train.head())
```

```
          Id  Product_Info_1 Product_Info_2  Product_Info_3  Product_Info_4  \
34822  46261               1             A8              26        0.230769
8612   11479               1             A8              26        0.372308
42717  56784               1             A1              26        0.230769
45825  60966               1             D4              26        0.230769
30524  40558               1             D4              26        0.230769

       Product_Info_5  Product_Info_6  Product_Info_7   Ins_Age        Ht  \
34822               2               3               1  0.626866  0.636364
8612                2               3               1  0.611940  0.800000
42717               2               3               1  0.298507  0.781818
45825               2               3               1  0.164179  0.727273
30524               2               3               1  0.074627  0.672727

       ...  Medical_Keyword_40  Medical_Keyword_41  Medical_Keyword_42  \
34822  ...                   0                   0                   0
8612   ...                   0                   0                   0
42717  ...                   0                   0                   0
45825  ...                   0                   0                   0
30524  ...                   0                   0                   0
```

```
       Medical_Keyword_43  Medical_Keyword_44  Medical_Keyword_45  \
34822                   0                   0                   0
8612                    0                   0                   0
42717                   0                   0                   0
45825                   0                   0                   0
30524                   0                   0                   0

       Medical_Keyword_46  Medical_Keyword_47  Medical_Keyword_48  Response
34822                   0                   0                   0         7
8612                    0                   0                   0         8
42717                   0                   0                   0         5
45825                   0                   0                   0         8
30524                   0                   0                   0         8

[5 rows x 128 columns]
```

Вычисление средних значений для каждой оценки

Проведем кластеризацию данных: разобьем их на группы по известной оценку скоринга (Response), вычислим центры этих групп как средние значения биометрических параметров.

Ввод [4]:
```
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
responses = np.arange(1, data["Response"].max() + 1)
clusters = [{}]*(len(responses) + 1)
for r in responses:
    clusters[r] = {}
    for c in columns:
        clusters[r][c] = data[data["Response"] == r][c].median()
print (clusters)
```

```
[{}, {'Wt': 0.309623431, 'Ht': 0.709090909, 'Ins_Age': 0.52238806, 'BMI': 0.483173533}, {'Wt': 0.330543933, 'Ht': 0.727272727, 'Ins_Age': 0.47761194, 'BMI': 0.510582282}, {'Wt': 0.31799163199999997, 'Ht': 0.727272727, 'Ins_Age': 0.358208955, 'BMI': 0.510582282}, {'Wt': 0.257322176, 'Ht': 0.709090909, 'Ins_Age': 0.328358209, 'BMI': 0.4196545055}, {'Wt': 0.351464435, 'Ht': 0.709090909, 'Ins_Age': 0.402985075, 'BMI': 0.5918224864999999}, {'Wt': 0.309623431, 'Ht': 0.727272727, 'Ins_Age': 0.432835821, 'BMI': 0.489742491}, {'Wt': 0.290794979, 'Ht': 0.709090909, 'Ins_Age': 0.447761194, 'BMI': 0.4707726829999997}, {'Wt': 0.23640167399999998, 'Ht': 0.690909091, 'Ins_Age': 0.313432836, 'BMI': 0.39487456299999996}]
```

Выполним предсказание оценки скоринга на основе средних значений

Будем использовать евклидово расстояние:

$$D = \sqrt{\sum (a_i - C_i)^2},$$

где a_i – значение параметров в проверочной выборке;

C_i – значение центров кластеров по данным обучающей выборки.

Выберем принадлежность к кластеру, расстояние до которого минимально:

Ввод [5]:
```
def calc_model (x):
    D_min = 10000000
    target = 0
    for _, cluster in enumerate(clusters):
        if len(cluster) > 0:
            D = 0
            for c in columns:
                D += (cluster[c] - x[c])**2
            D = np.sqrt(D)
            if D < D_min:
                target = _
                D_min = D
```

```
    x["target"] = target
    x["random"] = int(np.random.uniform(1, 8.01, 1)[0])
    x["sample"] = data.sample(1)["Response"].values[0]
    x["all8"] = 8
    return x
```

Ввод [6]:
```
data_test = data_test.apply(calc_model, axis=1, result_type="expand")
print (data_test.head(20))
```

```
          Id  Product_Info_1 Product_Info_2  Product_Info_3  Product_Info_4  \
51552  68645               1             B2              26        1.000000
46973  62508               1             E1              10        0.179487
7294    9732               1             D4              10        0.487179
16815  22403               1             D3              26        0.076923
34127  45300               1             D3              26        0.076923
1765    2375               1             A6              26        0.128205
48026  63957               1             D1              26        0.230769
22911  30549               1             D3              26        0.333333
34641  46016               1             A8              26        0.230769
37132  49302               1             E1              10        0.097608
26183  34861               1             C1              26        0.282051
41440  55061               1             D2              26        1.000000
56860  75754               1             A6              26        0.297436
33724  44773               1             E1              26        0.341915
3699    4926               2             A3              26        0.128205
10702  14216               1             D4              26        0.230769
2220    2972               1             D3              26        0.230769
25795  34352               1             D1              26        0.076923
53993  71896               1             D2              26        0.743590
21488  28606               1             A8              26        0.266201

       Product_Info_5  Product_Info_6  Product_Info_7   Ins_Age        Ht  \
51552               2               3               1  0.149254  0.672727
46973               2               3               1  0.701493  0.654545
7294                2               3               1  0.343284  0.618182
16815               2               3               1  0.567164  0.690909
34127               2               3               1  0.671642  0.654545
1765                2               3               1  0.298507  0.618182
48026               2               1               1  0.761194  0.763636
22911               2               3               1  0.179104  0.727273
34641               2               3               1  0.477612  0.781818
37132               2               3               1  0.507463  0.709091
26183               2               1               1  0.656716  0.781818
41440               2               3               1  0.373134  0.854545
56860               2               1               1  0.283582  0.690909
33724               2               3               1  0.477612  0.581818
3699                2               3               1  0.134328  0.672727
10702               2               3               1  0.373134  0.600000
2220                2               1               1  0.507463  0.636364
25795               2               3               1  0.776119  0.818182
53993               2               3               1  0.507463  0.800000
21488               2               1               1  0.671642  0.781818

       ...  Medical_Keyword_44  Medical_Keyword_45  Medical_Keyword_46  \
51552  ...                   0                   0                   0
46973  ...                   0                   0                   0
7294   ...                   0                   0                   0
16815  ...                   0                   0                   0
34127  ...                   0                   0                   0
1765   ...                   0                   0                   0
48026  ...                   0                   0                   0
```

```
22911  ...                    0                    0                    0
34641  ...                    0                    0                    0
37132  ...                    0                    0                    0
26183  ...                    0                    0                    0
41440  ...                    0                    0                    0
56860  ...                    0                    0                    0
33724  ...                    0                    0                    0
3699   ...                    0                    0                    0
10702  ...                    0                    0                    0
2220   ...                    0                    0                    0
25795  ...                    0                    0                    0
53993  ...                    0                    0                    0
21488  ...                    0                    0                    0

       Medical_Keyword_47  Medical_Keyword_48  Response  target  random  \
51552                   0                   0         7       8       1
46973                   0                   0         5       1       4
7294                    0                   0         8       8       6
16815                   0                   0         6       1       6
34127                   0                   0         7       1       7
1765                    0                   0         8       4       1
48026                   0                   0         5       1       5
22911                   0                   0         8       8       5
34641                   0                   0         8       2       4
37132                   0                   0         8       1       7
26183                   0                   0         6       1       3
41440                   1                   0         8       3       3
56860                   0                   0         6       3       3
33724                   0                   0         4       8       6
3699                    1                   0         8       8       7
10702                   0                   0         8       8       6
2220                    0                   0         8       7       3
25795                   0                   0         1       1       1
53993                   0                   0         8       1       2
21488                   0                   0         1       1       3

       sample  all8
51552       1     8
46973       6     8
7294        7     8
16815       1     8
34127       5     8
1765        6     8
48026       2     8
22911       8     8
34641       8     8
37132       7     8
26183       6     8
41440       5     8
56860       7     8
33724       6     8
3699        8     8
10702       8     8
2220        7     8
25795       7     8
53993       7     8
21488       2     8

[20 rows x 132 columns]
```

Оценка качества модели: F1

скоринг \ исходные данные	8	1
8	TP	FP
1	FN	TN

$$\text{Точность} = \frac{TP}{TP + FP}$$

$$\text{Полнота} = \frac{TP}{TP + FN}$$

$$F1 = 2 * \frac{\text{Точность} * \text{Полнота}}{\text{Точность} + \text{Полнота}}$$

Ввод [7]:
```
print ("Случайный выбор:",
       f1_score(data_test["random"], data_test["Response"],
                average="weighted"))
print ("Выбор по частоте:",
       f1_score(data_test["sample"], data_test["Response"],
                average="weighted"))
print ("Кластеризация:",
       f1_score(data_test["target"], data_test["Response"],
                average="weighted"))
print ("Самый популярный:",
       f1_score(data_test["all8"], data_test["Response"],
                average="weighted"))
```

```
Случайный выбор: 0.10557781054202212
Выбор по частоте: 0.19234494336132174
Кластеризация: 0.26757674537056736
Самый популярный: 0.49496293480326936
```

Матрица неточностей (Confusion Matrix)

скоринг \ исходные данные	1	2	3	4	5	6	7	8
1	TP	FP1	FP2	FP3	FP4	FP5	FP6	FP7
2	FN1	TP	FP1	FP2	FP3	FP4	FP5	FP6
3	FN2	FN1	TP	FP1	FP2	FP3	FP4	FP5
4	FN3	FN2	FN1	TP	FP1	FP2	FP3	FP4
5	FN4	FN3	FN2	FN1	TP	FP1	FP2	FP3
6	FN5	FN4	FN3	FN2	FN1	TP	FP1	FP2
7	FN6	FN5	FN4	FN3	FN2	FN1	TP	FP1
8	FN7	FN6	FN5	FN4	FN3	FN2	FN1	TP

```
Ввод [9]: print (confusion_matrix(data_test["target"], data_test["Response"]))
```

```
[[ 544  472   39   53  268  657  601  833]
 [ 101   81   14   12   72  188  116   65]
 [  91  132   23   36  147  369  249  194]
 [  59   61   17   47   40  235  154  519]
 [ 214  330   50    2  450  254   30    6]
 [  22   22    5    7   16   72   50   72]
 [  67   79    9   25   24  125  127  346]
 [ 160  127   32  124   98  312  260 1871]]
```

Квадратичный коэффициент каппа Коэна

Является логичным продолжением оценке по матрице неточностей, но более точно указывает на соответствие вычисленных значений реальным, поскольку используем матрицу весов: большая ошибка получает больший вес.

Для расчета требуется вычислить матрицу весов (W), 8 × 8 выглядит так (каждый элемент – это квадрат разницы между номером строки и номером столба, разделенный на 64):

матрица весов	1	2	3	4	5	6	7	8
1	0	0.015625	0.0625	0.140625	0.25	0.390625	0.5625	0.765625
2	0.015625	0	0.015625	0.0625	0.140625	0.25	0.390625	0.5625
3	0.0625	0.015625	0	0.015625	0.0625	0.25	0.390625	0.5625
4	0.140625	0.0625	0.015625	0	0.015625	0.0625	0.25	0.390625
5	0.25	0.140625	0.0625	0.015625	0	0.015625	0.0625	0.25
6	0.390625	0.25	0.140625	0.0625	0.015625	0	0.015625	0.0625
7	0.5625	0.390625	0.25	0.140625	0.0625	0.015625	0	0.015625
8	0.765625	0.5625	0.390625	0.25	0.140625	0.0625	0.015625	0

После вычисления матрицы неточностей (O) вычисляют матрицу гистограмм расчетных и идеальных значений (E) – сколько всего оценок 1, оценок 2, и т. д. В случае оценок от 1 до 8 гистограммы будут выглядеть следующим образом:

Расчет: [3372, 661, 1244, 1040, 1380, 276, 900, 3004]
Идеал: [1193, 1302, 207, 261, 1120, 2257, 1633, 3904]

Каждый элемент матрицы *ij* – это произведение *i*-расчетного значения на *j*-идеальное. Например, для ячейки 1-1 это будет 3372 * 1193 = 4022796. И так далее.

Матрицу неточностей и матрицу гистограмм нормируют (делят каждый элемент матрицы на сумму всех элементов) и вычисляют взвешенную сумму, используя матрицу весов (каждый элемент матрицы весов умножают на соответствующий элемент другой матрицы, все произведения суммируют): $e = W * E$, $o = W * O$.

Значение Kappa (каппа) вычисляется как $1 - o/e$.

```
Ввод [10]: print (cohen_kappa_score(data_test["target"], data_test["Response"],
                  weights="quadratic"))
           print (cohen_kappa_score(data_test["all8"], data_test["Response"],
                  weights="quadratic"))
```

```
0.19375052727446362
0.0
```

МЕТОД БЛИЖАЙШИХ СОСЕДЕЙ

Постановка задачи

Загрузим данные и разделим выборку на обучающую/проверочную в соотношении 80/20.

Применим метод ближайших соседей (kNN) для классификации скоринга. Будем использовать только биометрические данные.

Проверим качество предсказания через каппа-метрику и матрицу неточностей.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

```
Ввод [1]: import pandas as pd
          import numpy as np
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import cohen_kappa_score, confusion_matrix
          from sklearn.neighbors import KNeighborsClassifier
```

Загрузка данных

```
Ввод [2]: data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
          print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Разделение данных

```
Ввод [3]: data_train, data_test = train_test_split(data, test_size=0.2)
          print (data_train.head())
```

	Id	Product_Info_1	Product_Info_2	Product_Info_3	Product_Info_4	\
1871	2505	1	D4	26	0.230769	
17271	23020	1	A1	26	0.076923	
2981	3972	1	D4	10	0.230769	
50679	67485	1	D3	26	1.000000	
4969	6612	1	D3	26	0.698247	

```
       Product_Info_5  Product_Info_6  Product_Info_7   Ins_Age        Ht  \
1871                2               3               1  0.432836  0.600000
17271               2               3               1  0.552239  0.672727
2981                2               3               1  0.283582  0.636364
50679               2               3               1  0.417910  0.781818
4969                2               3               1  0.388060  0.672727

       ...  Medical_Keyword_40  Medical_Keyword_41  Medical_Keyword_42  \
1871   ...                   0                   0                   0
17271  ...                   0                   0                   0
2981   ...                   0                   0                   1
50679  ...                   0                   0                   0
4969   ...                   0                   0                   0

       Medical_Keyword_43  Medical_Keyword_44  Medical_Keyword_45  \
1871                    0                   0                   0
17271                   0                   0                   0
2981                    0                   0                   0
50679                   0                   0                   0
4969                    0                   0                   0

       Medical_Keyword_46  Medical_Keyword_47  Medical_Keyword_48  Response
1871                    0                   0                   0         8
17271                   0                   0                   0         2
2981                    0                   0                   0         8
50679                   0                   0                   0         8
4969                    0                   0                   0         8

[5 rows x 128 columns]
```

Расчет модели kNN (k ближайших соседей)

Вычисляем не центры (кластеры) исходных групп, а расстояние до всех значений. Выбираем то значение, которое превалирует у k ближайших соседей.

Для оценки качества модели возьмем k равным 10, 100, 1000, 10000.

Ввод [4]:
```
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
max_nn = data_train.groupby("Response").count()["Id"].min()
knn10 = KNeighborsClassifier(n_neighbors=10)
knn100 = KNeighborsClassifier(n_neighbors=100)
knn1000 = KNeighborsClassifier(n_neighbors=1000)
knn10000 = KNeighborsClassifier(n_neighbors=10000)
knnmax = KNeighborsClassifier(n_neighbors=max_nn)
```

Ввод [6]:
```
y = data_train["Response"]
x = pd.DataFrame(data_train, columns=columns)
knn10.fit(x, y)
knn100.fit(x, y)
knn1000.fit(x, y)
knn10000.fit(x, y)
knnmax.fit(x, y)
```

Out[6]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=801, p=2,
                     weights='uniform')
```

Предсказание данных

Внимание: 10000 соседей потребует порядка 4 Гб оперативной памяти

Ввод [7]:
```
data_test = pd.DataFrame(data_test)
x_test = pd.DataFrame(data_test, columns=columns)
data_test["target_10"] = knn10.predict(x_test)
data_test["target_100"] = knn100.predict(x_test)
data_test["target_1000"] = knn1000.predict(x_test)
data_test["target_10000"] = knn10000.predict(x_test)
data_test["target_max"] = knnmax.predict(x_test)
print (data_test.head(20))
```

```
          Id  Product_Info_1 Product_Info_2  Product_Info_3  Product_Info_4  \
7633   10172               1             E1              10        0.128205
38548  51180               1             D3              26        0.487179
53431  71135               1             D2              29        0.076923
18262  24353               1             D1              26        0.230769
11147  14794               1             D3              26        0.282051
58236  77622               1             D2              26        0.384615
37339  49586               1             D3              10        0.128205
25533  34002               1             D2              26        0.487179
11295  14987               1             D4              26        0.487179
11842  15714               1             D3              26        0.487179
12528  16653               1             A8              10        0.076923
34853  46298               1             B2              26        0.230769
4853    6463               1             D2              26        0.487179
33387  44325               1             A8              26        0.025641
41926  55690               1             D3              26        0.230769
30654  40724               1             D4              26        0.102564
37750  50123               1             A5              26        0.076923
13038  17309               1             D4              26        0.282051
18659  24897               1             D4              26        0.487179
54369  72406               1             A7              26        0.006154

       Product_Info_5  Product_Info_6  Product_Info_7   Ins_Age        Ht  \
7633                2               3               1  0.820896  0.600000
38548               2               3               1  0.388060  0.745455
53431               2               1               1  0.686567  0.745455
18262               2               3               1  0.582090  0.781818
11147               2               3               1  0.492537  0.600000
58236               2               1               1  0.462687  0.709091
37339               2               3               1  0.567164  0.636364
25533               2               1               1  0.597015  0.600000
11295               2               3               1  0.164179  0.672727
11842               2               3               1  0.701493  0.709091
12528               2               3               1  0.447761  0.563636
34853               2               3               1  0.223881  0.690909
4853                2               3               1  0.462687  0.818182
33387               2               3               1  0.641791  0.763636
41926               2               1               1  0.477612  0.636364
30654               2               3               1  0.268657  0.654545
37750               2               3               1  0.059701  0.600000
13038               2               3               1  0.194030  0.727273
18659               2               3               1  0.268657  0.709091
54369               2               3               1  0.761194  0.690909

       ...  Medical_Keyword_45  Medical_Keyword_46  Medical_Keyword_47  \
7633   ...                   0                   0                   0
38548  ...                   0                   0                   0
53431  ...                   0                   0                   0
18262  ...                   0                   0                   0
11147  ...                   0                   0                   0
58236  ...                   0                   0                   0
37339  ...                   1                   0                   0
25533  ...                   0                   0                   0
11295  ...                   0                   0                   0
11842  ...                   0                   0                   0
12528  ...                   0                   0                   0
34853  ...                   0                   0                   0
4853   ...                   0                   0                   1
33387  ...                   0                   0                   0
41926  ...                   0                   0                   0
```

```
41926  ...                   0                 0                 0
30654  ...                   0                 0                 0
37750  ...                   0                 0                 0
13038  ...                   0                 0                 0
18659  ...                   0                 0                 0
54369  ...                   0                 0                 0
```

	Medical_Keyword_48	Response	target_10	target_100	target_1000	\
7633	1	1	2	2	1	
38548	0	7	6	6	6	
53431	1	2	2	2	2	
18262	0	7	2	8	8	
11147	0	6	6	5	5	
58236	0	1	6	8	8	
37339	0	6	8	8	8	
25533	0	8	5	8	8	
11295	0	8	8	8	8	
11842	0	2	2	8	8	
12528	0	8	7	8	8	
34853	0	8	8	8	8	
4853	0	6	5	5	6	
33387	0	5	1	8	8	
41926	0	8	8	8	8	
30654	0	7	8	8	8	
37750	0	4	8	8	8	
13038	0	8	8	8	8	
18659	0	8	8	8	8	
54369	0	2	2	1	1	

	target_10000	target_max
7633	8	1
38548	6	6
53431	6	2
18262	6	7
11147	6	5
58236	8	8
37339	8	8
25533	8	8
11295	8	8
11842	8	8
12528	8	8
34853	8	8
4853	6	6
33387	8	8
41926	8	8
30654	8	8
37750	8	8
13038	8	8
18659	8	8
54369	6	1

Оценка модели

Ввод [8]:
```
print ("kNN, 10:",
       cohen_kappa_score(data_test["target_10"], data_test["Response"],
                         weights="quadratic"))
print ("kNN, 100:",
       cohen_kappa_score(data_test["target_100"], data_test["Response"],
                         weights="quadratic"))
print ("kNN, 1000:",
       cohen_kappa_score(data_test["target_1000"], data_test["Response"],
                         weights="quadratic"))
```

```
print ("kNN, 10000:",
       cohen_kappa_score(data_test["target_10000"], data_test["Response"],
                         weights="quadratic"))
print ("kNN, max:",
       cohen_kappa_score(data_test["target_max"], data_test["Response"],
                         weights="quadratic"))
```

```
kNN, 10: 0.2957062865969361
kNN, 100: 0.30326951710809413
kNN, 1000: 0.28175078115596264
kNN, 10000: 0.15030991250481207
kNN, max: 0.28390472176878834
```

Матрица неточностей

```
Ввод [9]: print (confusion_matrix(data_test["target_10"],
                                  data_test["Response"]))
          print (confusion_matrix(data_test["target_10000"],
                                  data_test["Response"]))
```

```
[[ 209  162   19    9   84  224  153  148]
 [ 153  244   20    8  142  166   97  112]
 [   1    1    0    0    0    3    0    1]
 [   0    2    0    1    1    4    2   12]
 [  96  163   45    0  345  153   38   29]
 [ 243  274   54   49  226  690  382  411]
 [ 112  116   16   38   70  276  270  269]
 [ 371  344   58  207  208  768  673 2905]]
[[   0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]
 [   1    1    0    0    0    0    0    0]
 [ 506  699  103   29  644  926  439  154]
 [   0    0    0    0    0    0    0    0]
 [ 678  606  109  283  432 1358 1176 3733]]
```

НАИВНЫЙ БАЙЕС В ЗАДАЧЕ КЛАССИФИКАЦИИ СКОРИНГА И ОПТИМИЗАЦИИ ПОТРЕБЛЕНИЯ ПАМЯТИ

Постановка задачи

Загрузим данные и подготовим все данные для анализа: проведем нормализацию и преобразование категорий. Оптимизируем потребление памяти.

Разделим выборку на обучающую/проверочную в соотношении 80/20.

Применим наивный Байес для классификации скоринга. Будем использовать все возможные столбцы.

Проверим качество предсказания через каппа-метрику и матрицу неточностей.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn import preprocessing
```

Загрузка данных

Ввод [2]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Категоризация данных

Ввод [3]:
```
data["Product_Info_2_1"] = data["Product_Info_2"].str.slice(0, 1)
data["Product_Info_2_2"] = pd.to_numeric(data["Product_Info_2"].str.slice(1, 2))
data.drop(labels=["Product_Info_2"], axis=1, inplace=True)
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 129 entries, Id to Product_Info_2_2
dtypes: float64(18), int64(110), object(1)
memory usage: 58.4+ MB
None
```

Оптимизация потребления памяти

Ввод [8]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
```

```
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print ("Потребление памяти меньше на", round(start_mem - end_mem, 2), "Мб (минус)",
           round(100*(start_mem - end_mem) / start_mem, 1), "%)")
    return df
```

Ввод [9]:
```
data = reduce_mem_usage(data)
print (data.info())
```

```
Потребление памяти меньше на 49.89 Мб (минус) 85.4 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 129 entries, Id to Product_Info_2_2
dtypes: category(1), float16(18), int16(1), int32(1), int8(108)
memory usage: 8.6 MB
None
```

Предобработка: категоризация, единичные векторы

Product
A
B
C
A

Переходит в

ProductA	ProductB	ProductC
1	0	0
0	1	0
0	0	1
1	0	0

Можно использовать sklearn.preprocessing.OneHotEncoder, но для этого потребуется дополнительно преобразовать фрейм данных (набор единичных векторов для каждого кортежа данных).

Также не будем использовать кодирование категорий (A->1, B->2, C->3, D->4, E->5), потому что это переводит номинативную случайную величину в ранговую/числовую, и является существенным допущением относительно исходных данных.

Ввод [10]:
```
for l in data["Product_Info_2_1"].unique():
    data["Product_Info_2_1" + l] = data["Product_Info_2_1"].isin([l]).astype("int8")
data.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
```

Заполним отсутствующие значения

-1 увеличивает "расстояние" при расчете ближайших соседей

Ввод [12]:
```
data.fillna(value=-1, inplace=True)
```

Столбцы для модели

Ввод [13]:
```
columns_groups = ["Insurance_History", "InsuredInfo", "Medical_Keyword",
                  "Family_Hist", "Medical_History", "Product_Info"]
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
for cg in columns_groups:
    columns.extend(data.columns[data.columns.str.startswith(cg)])
print (columns)
```

```
['Wt', 'Ht', 'Ins_Age', 'BMI', 'Insurance_History_1', 'Insurance_History_2', 'Insurance_History_3', 'Insurance_Histor
y_4', 'Insurance_History_5', 'Insurance_History_7', 'Insurance_History_8', 'Insurance_History_9', 'InsuredInfo_1', 'I
nsuredInfo_2', 'InsuredInfo_3', 'InsuredInfo_4', 'InsuredInfo_5', 'InsuredInfo_6', 'InsuredInfo_7', 'Medical_Keyword_
1', 'Medical_Keyword_2', 'Medical_Keyword_3', 'Medical_Keyword_4', 'Medical_Keyword_5', 'Medical_Keyword_6', 'Medical
_Keyword_7', 'Medical_Keyword_8', 'Medical_Keyword_9', 'Medical_Keyword_10', 'Medical_Keyword_11', 'Medical_Keyword_1
2', 'Medical_Keyword_13', 'Medical_Keyword_14', 'Medical_Keyword_15', 'Medical_Keyword_16', 'Medical_Keyword_17', 'Me
dical_Keyword_18', 'Medical_Keyword_19', 'Medical_Keyword_20', 'Medical_Keyword_21', 'Medical_Keyword_22', 'Medical_K
eyword_23', 'Medical_Keyword_24', 'Medical_Keyword_25', 'Medical_Keyword_26', 'Medical_Keyword_27', 'Medical_Keyword_
28', 'Medical_Keyword_29', 'Medical_Keyword_30', 'Medical_Keyword_31', 'Medical_Keyword_32', 'Medical_Keyword_33', 'M
edical_Keyword_34', 'Medical_Keyword_35', 'Medical_Keyword_36', 'Medical_Keyword_37', 'Medical_Keyword_38', 'Medical_
Keyword_39', 'Medical_Keyword_40', 'Medical_Keyword_41', 'Medical_Keyword_42', 'Medical_Keyword_43', 'Medical_Keyword
_44', 'Medical_Keyword_45', 'Medical_Keyword_46', 'Medical_Keyword_47', 'Medical_Keyword_48', 'Family_Hist_1', 'Famil
y_Hist_2', 'Family_Hist_3', 'Family_Hist_4', 'Family_Hist_5', 'Medical_History_1', 'Medical_History_2', 'Medical_Hist
ory_3', 'Medical_History_4', 'Medical_History_5', 'Medical_History_6', 'Medical_History_7', 'Medical_History_8', 'Med
ical_History_9', 'Medical_History_10', 'Medical_History_11', 'Medical_History_12', 'Medical_History_13', 'Medical_His
tory_14', 'Medical_History_15', 'Medical_History_16', 'Medical_History_17', 'Medical_History_18', 'Medical_History_19
', 'Medical_History_20', 'Medical_History_21', 'Medical_History_22', 'Medical_History_23', 'Medical_History_24', 'Med
ical_History_25', 'Medical_History_26', 'Medical_History_27', 'Medical_History_28', 'Medical_History_29', 'Medical_Hi
story_30', 'Medical_History_31', 'Medical_History_32', 'Medical_History_33', 'Medical_History_34', 'Medical_History_3
5', 'Medical_History_36', 'Medical_History_37', 'Medical_History_38', 'Medical_History_39', 'Medical_History_40', 'Me
dical_History_41', 'Product_Info_1', 'Product_Info_3', 'Product_Info_4', 'Product_Info_5', 'Product_Info_6', 'Product
_Info_7', 'Product_Info_2_2', 'Product_Info_2_1D', 'Product_Info_2_1A', 'Product_Info_2_1E', 'Product_Info_2_1C', 'Pr
oduct_Info_2_1B']
```

Предобработка данных

Дополнительно проведем z-нормализацию данных через предварительную обработку (preprocessing). Нормализуем весь исходный набор данных.

Ввод [14]:
```
scaler = preprocessing.StandardScaler()
scaler.fit(pd.DataFrame(data, columns=columns))
```

Out[14]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

Разделение данных

Ввод [15]:
```
data_train, data_test = train_test_split(data, test_size=0.2)
print (data_train.head())
```

```
          Id  Product_Info_1  Product_Info_3  Product_Info_4  Product_Info_5  \
51154  68123               1              26        0.076904               2
37067  49216               1              29        0.487061               2
11378  15091               1              26        0.076904               2
5068    6749               1              37        0.076904               2
34965  46443               1              26        0.128174               2

       Product_Info_6  Product_Info_7   Ins_Age        Ht        Wt  ...  \
51154               3               1  0.313477  0.781738  0.592285  ...
37067               3               1  0.193970  0.763672  0.435059  ...
11378               3               1  0.761230  0.545410  0.194580  ...
5068                3               1  0.029846  0.618164  0.236450  ...
34965               3               1  0.522461  0.763672  0.299072  ...

       Medical_Keyword_46  Medical_Keyword_47  Medical_Keyword_48  Response  \
51154                   0                   0                   1         2
37067                   0                   0                   0         5
11378                   0                   0                   0         5
5068                    0                   0                   0         8
34965                   0                   0                   1         8

       Product_Info_2_2  Product_Info_2_1D  Product_Info_2_1A  \
51154                 3                  1                  0
37067                 4                  1                  0
11378                 2                  1                  0
5068                  3                  1                  0
34965                 8                  0                  1
```

```
       Product_Info_2_1E  Product_Info_2_1C  Product_Info_2_1B
51154                  0                  0                  0
37067                  0                  0                  0
11378                  0                  0                  0
5068                   0                  0                  0
34965                  0                  0                  0

[5 rows x 133 columns]
```

Расчет модели наивного Байеса

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Для каждого параметра вычисляется его вероятность принять определенное значение - P(B). Для каждого класса вычисляется его вероятность (по факту, доля) - P(A). Затем вычисляется вероятность для каждого параметра принять определенное значение при определенном классе - P(B\A).

По всем вычисленным значениям находится вероятность при известных параметрах принять какое-либо значение класса.

Ввод [18]:
```
y = data_train["Response"]
x = scaler.transform(pd.DataFrame(data_train, columns=columns))
```

Ввод [19]:
```
bayes = GaussianNB()
bayes.fit(x, y)
```

Out[19]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Предсказание данных

Ввод [20]:
```
data_test = pd.DataFrame(data_test)
x_test = scaler.transform(pd.DataFrame(data_test, columns=columns))
data_test["target"] = bayes.predict(x_test)
print (data_test.head())
```

```
           Id  Product_Info_1  Product_Info_3  Product_Info_4  Product_Info_5  \
12178  16185               1              26        0.154907               2
29527  39272               1              26        1.000000               2
8574   11422               1              15        0.230713               2
27890  37150               1              26        0.230713               2
8851   11793               2              26        1.000000               2

       Product_Info_6  Product_Info_7   Ins_Age        Ht        Wt  ...  \
12178               3               1  0.626953  0.799805  0.386963  ...
29527               3               1  0.253662  0.836426  0.351562  ...
8574                3               1  0.283691  0.745605  0.320068  ...
27890               3               1  0.223877  0.745605  0.267822  ...
8851                1               1  0.522461  0.708984  0.414307  ...

       Medical_Keyword_47  Medical_Keyword_48  Response  Product_Info_2_2  \
12178                   0                   0         8                 1
29527                   0                   0         4                 3
8574                    0                   0         7                 4
27890                   0                   0         8                 4
8851                    0                   0         2                 1

       Product_Info_2_1D  Product_Info_2_1A  Product_Info_2_1E  \
12178                  1                  0                  0
29527                  1                  0                  0
8574                   1                  0                  0
27890                  1                  0                  0
8851                   1                  0                  0

       Product_Info_2_1C  Product_Info_2_1B  target
12178                  0                  0       4
29527                  0                  0       4
8574                   0                  0       4
27890                  0                  0       4
8851                   0                  0       2

[5 rows x 134 columns]
```

Оценка модели

Ввод [21]:
```
print ("Байес:", cohen_kappa_score(data_test["target"],
                data_test["Response"], weights="quadratic"))
```

```
Байес: 0.1993841553537673
```

Матрица неточностей

Ввод [22]:
```
print (confusion_matrix(data_test["target"], data_test["Response"]))
```

```
[[ 289  212   10   12   93  178   79  112]
 [ 110  170    3    3   52   79   32   35]
 [ 102  115   17    8   58   85   19   35]
 [ 317  338  115  233  470 1021  700 2149]
 [  44   75    2    1   26   34    8   10]
 [  13   19    0    1    4   17    2    3]
 [ 340  376   32   43  396  739  678  812]
 [  44   20    5    8   32   76   94  672]]
```

ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

Постановка задачи

Загрузим данные, приведем их к числовым, заполним пропуски, нормализуем данные и оптимизируем память.

Разделим выборку на обучающую/проверочную в соотношении 80/20.

Применим логистическую регрессию по всему набору данных.

Проведем предсказание и проверим качество через каппа-метрику.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [2]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
```

Загрузка данных

Ввод [3]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Предобработка данных

Ввод [4]:
```
data["Product_Info_2_1"] = data["Product_Info_2"].str.slice(0, 1)
data["Product_Info_2_2"] = pd.to_numeric(data["Product_Info_2"].str.slice(1, 2))
data.drop(labels=["Product_Info_2"], axis=1, inplace=True)
for l in data["Product_Info_2_1"].unique():
    data["Product_Info_2_1" + l] = data["Product_Info_2_1"].isin([l]).astype("int8")
data.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
data.fillna(value=-1, inplace=True)
```

Оптимизация памяти

Ввод [5]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус',
          round(100 * (start_mem - end_mem) / start_mem, 1), '%)')
    return df
```

Ввод [6]:
```
data = reduce_mem_usage(data)
print (data.info())
```

```
Потребление памяти меньше на 49.49 Мб (минус 84.9 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
```

```
Columns: 133 entries, Id to Product_Info_2_1B
dtypes: float16(18), int16(1), int32(1), int8(113)
memory usage: 8.8 MB
None
```

Набор столбцов для расчета

Ввод [7]:
```
columns_groups = ["Insurance_History", "InsuredInfo", "Medical_Keyword",
                  "Family_Hist", "Medical_History", "Product_Info"]
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
for cg in columns_groups:
    columns.extend(data.columns[data.columns.str.startswith(cg)])
print (columns)
```

```
['Wt', 'Ht', 'Ins_Age', 'BMI', 'Insurance_History_1', 'Insurance_History_2', 'Insurance_History_3', 'Insurance_Histor
y_4', 'Insurance_History_5', 'Insurance_History_7', 'Insurance_History_8', 'Insurance_History_9', 'Medical_Keyword_1
', 'Medical_Keyword_2', 'Medical_Keyword_3', 'Medical_Keyword_4', 'Medical_Keyword_5', 'Medical_Keyword_6', 'Medical_
Keyword_7', 'Medical_Keyword_8', 'Medical_Keyword_9', 'Medical_Keyword_10', 'Medical_Keyword_11', 'Medical_Keyword_12
', 'Medical_Keyword_13', 'Medical_Keyword_14', 'Medical_Keyword_15', 'Medical_Keyword_16', 'Medical_Keyword_17', 'Med
ical_Keyword_18', 'Medical_Keyword_19', 'Medical_Keyword_20', 'Medical_Keyword_21', 'Medical_Keyword_22', 'Medical_Ke
yword_23', 'Medical_Keyword_24', 'Medical_Keyword_25', 'Medical_Keyword_26', 'Medical_Keyword_27', 'Medical_Keyword_2
8', 'Medical_Keyword_29', 'Medical_Keyword_30', 'Medical_Keyword_31', 'Medical_Keyword_32', 'Medical_Keyword_33', 'Me
dical_Keyword_34', 'Medical_Keyword_35', 'Medical_Keyword_36', 'Medical_Keyword_37', 'Medical_Keyword_38', 'Medical_K
eyword_39', 'Medical_Keyword_40', 'Medical_Keyword_41', 'Medical_Keyword_42', 'Medical_Keyword_43', 'Medical_Keyword_
44', 'Medical_Keyword_45', 'Medical_Keyword_46', 'Medical_Keyword_47', 'Medical_Keyword_48', 'Family_Hist_1', 'Family
_Hist_2', 'Family_Hist_3', 'Family_Hist_4', 'Family_Hist_5', 'Medical_History_1', 'Medical_History_2', 'Medical_Histo
ry_3', 'Medical_History_4', 'Medical_History_5', 'Medical_History_6', 'Medical_History_7', 'Medical_History_8', 'Medi
cal_History_9', 'Medical_History_10', 'Medical_History_11', 'Medical_History_12', 'Medical_History_13', 'Medical_Hist
ory_14', 'Medical_History_15', 'Medical_History_16', 'Medical_History_17', 'Medical_History_18', 'Medical_History_19
', 'Medical_History_20', 'Medical_History_21', 'Medical_History_22', 'Medical_History_23', 'Medical_History_24', 'Med
ical_History_25', 'Medical_History_26', 'Medical_History_27', 'Medical_History_28', 'Medical_History_29', 'Medical_Hi
story_30', 'Medical_History_31', 'Medical_History_32', 'Medical_History_33', 'Medical_History_34', 'Medical_History_3
5', 'Medical_History_36', 'Medical_History_37', 'Medical_History_38', 'Medical_History_39', 'Medical_History_40', 'Me
dical_History_41', 'Product_Info_1', 'Product_Info_3', 'Product_Info_4', 'Product_Info_5', 'Product_Info_6', 'Product
_Info_7', 'Product_Info_2_2', 'Product_Info_2_1D', 'Product_Info_2_1A', 'Product_Info_2_1E', 'Product_Info_2_1C', 'Pr
oduct_Info_2_1B']
```

Предобработка данных

Дополнительно проведем z-нормализацию данных через предварительную обработку (preprocessing).

Ввод [8]:
```
scaler = preprocessing.StandardScaler()
scaler.fit(pd.DataFrame(data, columns=columns))
```

Out[8]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

Разделение данных

Преобразуем выборки в отдельные наборы данных

Ввод [12]:
```
data_train, data_test = train_test_split(data, test_size=0.2)
data_train = pd.DataFrame(data_train)
data_test = pd.DataFrame(data_test)
print (data_train.head())
```

```
          Id  Product_Info_1  Product_Info_3  Product_Info_4  Product_Info_5  \
26429  35215               1              26        0.076904               2
53967  71857               1              10        0.179443               2
4481    5963               1              26        0.146729               2
52422  69788               1              26        0.076904               2
18041  24047               1              26        0.487061               2

       Product_Info_6  Product_Info_7   Ins_Age        Ht        Wt  ...  \
26429               3               1  0.597168  0.672852  0.246826  ...
53967               3               1  0.313477  0.672852  0.253174  ...
4481                3               1  0.865723  0.600098  0.194580  ...
52422               3               1  0.462646  0.745605  0.299072  ...
18041               1               1  0.283691  0.672852  0.215454  ...
```

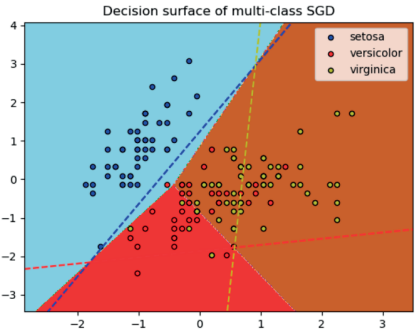
```
       Medical_Keyword_46  Medical_Keyword_47  Medical_Keyword_48  Response  \
26429                   0                   0                   0         8
53967                   0                   0                   0         7
4481                    0                   0                   0         2
52422                   0                   0                   0         6
18041                   0                   0                   0         8

       Product_Info_2_2  Product_Info_2_1D  Product_Info_2_1A  \
26429                 8                  0                  1
53967                 3                  0                  1
4481                  1                  0                  1
52422                 4                  1                  0
18041                 6                  0                  1

       Product_Info_2_1E  Product_Info_2_1C  Product_Info_2_1B
26429                  0                  0                  0
53967                  0                  0                  0
4481                   0                  0                  0
52422                  0                  0                  0
18041                  0                  0                  0

[5 rows x 133 columns]
```

Логистическая регрессия

$$P = \frac{exp^T}{1 + exp^T}$$

$$T = a_0 + b_1 x_1 + \cdots + b_n x_n$$

T - терминатор, логистическая кривая



Ввод [10]:

```
def regression_model (df, columns):
    y = df["Response"]
    x = scaler.transform(pd.DataFrame(df, columns=columns))
    model = LogisticRegression(max_iter=1000, class_weight='balanced',
                               multi_class='multinomial')
    model.fit(x, y)
    return model

def logistic_regression (columns):
    x = scaler.transform(pd.DataFrame(data_test, columns=columns))
    model = regression_model(data_train, columns)
    data_test["target"] = model.predict(x)
    return cohen_kappa_score(data_test["target"],
                             data_test["Response"], weights="quadratic")
```

Предсказание данных и оценка модели

Кластеризация дает 0.192, kNN(100) - 0.3

Ввод [13]:
```
print ("Логистическая регрессия:",
       round(logistic_regression(columns), 3))
```

```
Логистическая регрессия: 0.497
```

В соревновании на Kaggle 0.512 - **2248 место**

Матрица неточностей

Ввод [14]:
```
print (confusion_matrix(data_test["target"], data_test["Response"]))
```

```
[[ 315  221    7    3   76  190  109   96]
 [ 178  305    8    3  113  187   77   70]
 [ 113  131  104   63  115  286   36   20]
 [  75   61   50  172   38  373   63  123]
 [ 144  230    9    3  473  295  176  140]
 [  49   61   10   11   61  302  101  161]
 [ 146  127    3    8  110  333  614  380]
 [ 179  172    5   27  110  293  414 2949]]
```

ИЕРАРХИЯ ЛОГИСТИЧЕСКОЙ РЕГРЕССИИ

Постановка задачи

Загрузим данные, приведем их к числовым, заполним пропуски, нормализуем данные и оптимизируем память.

Разделим выборку на обучающую/проверочную в соотношении 80/20.

Построим 4 модели логистической регрессии: для 8, 6 и остальных классов, для 2, 5 и остальных, для 1, 7 и остальных, и для 4 и 3 – по убыванию частоты значения. Будем использовать перекрестную проверку при принятии решения об оптимальном наборе столбцов.

Проведем предсказание и проверим качество через каппа-метрику.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix, make_scorer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn import preprocessing
```

Загрузка данных

Ввод [2]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Предобработка данных

Ввод [3]:
```
data["Product_Info_2_1"] = data["Product_Info_2"].str.slice(0, 1)
data["Product_Info_2_2"] = pd.to_numeric(data["Product_Info_2"].str.slice(1, 2))
data.drop(labels=["Product_Info_2"], axis=1, inplace=True)
for l in data["Product_Info_2_1"].unique():
    data["Product_Info_2_1" + l] = data["Product_Info_2_1"].isin([l]).astype("int8")
data.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
data.fillna(value=-1, inplace=True)
```

Оптимизация памяти

Ввод [4]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус',
          round(100 * (start_mem - end_mem) / start_mem, 1), '%)')
    return df
```

Ввод [5]:
```
data = reduce_mem_usage(data)
print (data.info())
```

```
Потребление памяти меньше на 49.49 Мб (минус 84.9 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 133 entries, Id to Product_Info_2_1B
dtypes: float16(18), int16(1), int32(1), int8(113)
memory usage: 8.8 MB
None
```

Общий набор столбцов для расчета

Ввод [6]:
```
columns_groups = ["Insurance_History", "InsuredInfo", "Medical_Keyword",
                  "Family_Hist", "Medical_History", "Product_Info"]
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
for cg in columns_groups:
    columns.extend(data.columns[data.columns.str.startswith(cg)])
print (columns)
```

```
['Wt', 'Ht', 'Ins_Age', 'BMI', 'Insurance_History_1', 'Insurance_History_2', 'Insurance_History_3', 'Insurance_Histor
y_4', 'Insurance_History_5', 'Insurance_History_7', 'Insurance_History_8', 'Insurance_History_9', 'Medical_Keyword_1
', 'Medical_Keyword_2', 'Medical_Keyword_3', 'Medical_Keyword_4', 'Medical_Keyword_5', 'Medical_Keyword_6', 'Medical_
Keyword_7', 'Medical_Keyword_8', 'Medical_Keyword_9', 'Medical_Keyword_10', 'Medical_Keyword_11', 'Medical_Keyword_12
', 'Medical_Keyword_13', 'Medical_Keyword_14', 'Medical_Keyword_15', 'Medical_Keyword_16', 'Medical_Keyword_17', 'Med
ical_Keyword_18', 'Medical_Keyword_19', 'Medical_Keyword_20', 'Medical_Keyword_21', 'Medical_Keyword_22', 'Medical_Ke
yword_23', 'Medical_Keyword_24', 'Medical_Keyword_25', 'Medical_Keyword_26', 'Medical_Keyword_27', 'Medical_Keyword_2
8', 'Medical_Keyword_29', 'Medical_Keyword_30', 'Medical_Keyword_31', 'Medical_Keyword_32', 'Medical_Keyword_33', 'Me
dical_Keyword_34', 'Medical_Keyword_35', 'Medical_Keyword_36', 'Medical_Keyword_37', 'Medical_Keyword_38', 'Medical_K
eyword_39', 'Medical_Keyword_40', 'Medical_Keyword_41', 'Medical_Keyword_42', 'Medical_Keyword_43', 'Medical_Keyword_
44', 'Medical_Keyword_45', 'Medical_Keyword_46', 'Medical_Keyword_47', 'Medical_Keyword_48', 'Family_Hist_1', 'Family
_Hist_2', 'Family_Hist_3', 'Family_Hist_4', 'Family_Hist_5', 'Medical_History_1', 'Medical_History_2', 'Medical_Histo
ry_3', 'Medical_History_4', 'Medical_History_5', 'Medical_History_6', 'Medical_History_7', 'Medical_History_8', 'Medi
cal_History_9', 'Medical_History_10', 'Medical_History_11', 'Medical_History_12', 'Medical_History_13', 'Medical_Hist
ory_14', 'Medical_History_15', 'Medical_History_16', 'Medical_History_17', 'Medical_History_18', 'Medical_History_19
', 'Medical_History_20', 'Medical_History_21', 'Medical_History_22', 'Medical_History_23', 'Medical_History_24', 'Med
ical_History_25', 'Medical_History_26', 'Medical_History_27', 'Medical_History_28', 'Medical_History_29', 'Medical_Hi
story_30', 'Medical_History_31', 'Medical_History_32', 'Medical_History_33', 'Medical_History_34', 'Medical_History_3
5', 'Medical_History_36', 'Medical_History_37', 'Medical_History_38', 'Medical_History_39', 'Medical_History_40', 'Me
dical_History_41', 'Product_Info_1', 'Product_Info_3', 'Product_Info_4', 'Product_Info_5', 'Product_Info_6', 'Product
_Info_7', 'Product_Info_2_2', 'Product_Info_2_1D', 'Product_Info_2_1A', 'Product_Info_2_1E', 'Product_Info_2_1C', 'Pr
oduct_Info_2_1B']
```

Предобработка данных

Ввод [7]:
```
scaler = preprocessing.StandardScaler()
data_transformed = pd.DataFrame(scaler.fit_transform(pd.DataFrame(data,
                                                                  columns=columns)))
columns_transformed = data_transformed.columns
data_transformed["Response"] = data["Response"]
```

Разделение данных

Преобразуем выборки в отдельные наборы данных

Ввод [8]:
```
data_train, data_test = train_test_split(data_transformed,
                                         test_size=0.2)
data_train = pd.DataFrame(data_train)
data_test = pd.DataFrame(data_test)
print (data_train.head())
```

```
              0         1         2         3         4         5         6  \
17986 -0.514003 -1.200817  0.365036  0.154055  0.611857 -0.169414  0.862391
23658 -1.265313 -1.687657  0.289513 -0.651001 -1.634368 -0.169414  0.862391
4911  -0.678523 -1.444237  0.592844  0.084137 -1.634368 -0.169414  0.862391
47730 -0.514003 -0.957397 -0.921336 -0.015746  0.611857 -0.169414 -1.159587
11079  0.190693  0.022862  0.365036  0.257933  0.611857 -0.169414  0.862391

              7         8         9  ...       109       110       111  \
17986 -1.013721  0.861368 -0.928723  ... -0.083689  0.441621 -0.149284
23658 -1.013721  0.862242 -0.928723  ... -0.083689  0.441621 -0.149284
4911  -1.013721  0.874351 -0.928723  ... -0.083689 -2.264385 -0.149284
47730  1.101046 -1.156735  1.130555  ... -0.083689  0.441621 -0.149284
11079  0.043662  0.871657 -0.928723  ... -0.083689  0.441621 -0.149284

            112       113       114       115       116       117  Response
17986 -0.666860  0.750845 -0.623305 -0.216001 -0.128866 -0.142142         8
23658  2.134117 -1.331832  1.604350 -0.216001 -0.128866 -0.142142         7
4911  -0.666860  0.750845 -0.623305 -0.216001 -0.128866 -0.142142         6
47730 -0.200031  0.750845 -0.623305 -0.216001 -0.128866 -0.142142         7
11079 -0.666860  0.750845 -0.623305 -0.216001 -0.128866 -0.142142         6

[5 rows x 119 columns]
```

Логистическая регрессия

В обучающих данных пометим все классы, кроме 6 и 8, как 0 - и проведем обучение по такому набору данных.

Затем в оставшихся данных (в которых класс не равен 6 или 8) заменим все классы, кроме 7 и 1, на 0 - и снова проведем обучение. И т.д. Получим иерархию классификаторов: 8/6/нет -> 7/1/нет -> 2/5/нет -> 4/3

Ввод [20]:
```
def regression_model (columns, df):
    x = pd.DataFrame(df, columns=columns)
    model = LogisticRegression(max_iter=1000)
    model.fit(x, df["Response"])
    return model

def logistic_regression(columns, df_train):
    model = regression_model(columns, df_train)
    logr_grid = GridSearchCV(model, {}, cv=5, n_jobs=2,
                      scoring=make_scorer(cohen_kappa_score))
    x = pd.DataFrame(df_train, columns=columns)
    logr_grid.fit(x, df_train["Response"])
    return logr_grid.best_score_
```

Оптимальный набор столбцов

Для каждого уровня иерархии это будет свой набор столбцов в исходных данных.

Перекрестная проверка

Разбиваем обучающую выборку еще на k (часто 5) частей, на каждой части данных обучаем модель. Затем проверяем 1-ю, 2-ю, 3-ю, 4-ю части на 5; 1-ю, 2-ю, 3-ю, 5-ю части на 4 и т.д.

В итоге обучение пройдет весь набор данных, и каждая часть набора будет проверена на всех оставшихся (перекрестным образом).

Ввод [10]:
```
def find_opt_columns (data_train):
    kappa_score_opt = 0
    columns_opt = []
    for col in columns_transformed:
        kappa_score = logistic_regression([col], data_train)
        if kappa_score > kappa_score_opt:
            columns_opt = [col]
            kappa_score_opt = kappa_score
    for col in columns_transformed:
        if col not in columns_opt:
            columns_opt.append(col)
            kappa_score = logistic_regression(columns_opt, data_train)
            if kappa_score < kappa_score_opt:
                columns_opt.pop()
            else:
                kappa_score_opt = kappa_score
    return columns_opt, kappa_score_opt
```

Будем последовательно "урезать" набор данных при расчете более глубоких моделей: после получения разделения на 8 и остальные отсечем все данные со значением 8, и т.д.

После каждого расчета модели будем вычислять значения в проверочной выборке. Проверочную выборку нулями заполнять не будем, иначе оценка будет считаться некорректно.

Набор разделений 6/8, 2/5, 1/7, 3/4 дает наибольшую точность

Ввод [21]:
```
responses = [[6, 8], [2, 5], [1, 7], [3, 4]]
logr_models = [{}]*len(responses)
data_train_current = data_train.copy()
i = 0
for response in responses:
    m_train = data_train_current.copy()
    if response != [3,4]:
        m_train["Response"] = m_train["Response"].apply(lambda x:0 if x not in response else x)
    columns_opt, kappa_score_opt = find_opt_columns(m_train)
    print (i, kappa_score_opt, columns_opt)
    logr_models[i] = {
        "model": regression_model(columns_opt, m_train),
        "columns": columns_opt
    }
    if response != [3,4]:
        data_train_current = data_train_current[~data_train_current["Response"].isin(response)]
    i += 1
```

```
0 0.4213698840981315 [117, 0, 1, 2, 5, 6, 7, 8, 9, 12, 13, 14, 15, 18, 19, 23, 26, 27, 28, 30, 31, 34, 38, 43, 46, 4
7, 49, 51, 52, 53, 56, 57, 59, 60, 61, 63, 68, 69, 76, 78, 79, 81, 82, 84, 86, 87, 88, 90, 92, 94, 95, 99, 100, 102,
103, 104, 106, 107, 108, 110, 111, 113, 116]
1 0.18589479001417825 [69, 0, 1, 2, 3, 4, 5, 11, 12, 13, 14, 15, 16, 25, 26, 29, 31, 32, 34, 37, 38, 41, 42, 43, 45,
46, 47, 51, 59, 60, 61, 65, 67, 68, 70, 72, 73, 76, 79, 80, 81, 83, 84, 87, 88, 89, 90, 91, 92, 95, 97, 98, 100, 104,
107, 108, 112, 115, 116, 117]
2 0.539947145549091 [108, 0, 2, 3, 4, 5, 6, 8, 9, 12, 13, 14, 17, 20, 21, 23, 26, 27, 31, 32, 33, 34, 35, 36, 38, 39,
40, 44, 48, 49, 50, 51, 53, 54, 56, 61, 62, 65, 67, 68, 69, 70, 71, 73, 75, 77, 78, 79, 80, 81, 82, 83, 85, 86, 87, 8
8, 89, 90, 91, 92, 94, 95, 99, 101, 102, 104, 106, 107, 111, 112, 114, 115, 117]
3 0.45030542885452185 [114, 0, 1, 2, 3, 4, 5, 14, 17, 20, 23, 27, 29, 41, 45, 50, 54, 56, 57, 69, 72, 83, 91, 99, 10
1, 106]
```

Предсказание данных и оценка модели

Последовательно считаем предсказания для каждой классификации. После этого объединяем предсказание по иерархии.

Ввод [22]:
```
def logr_hierarchy (x):
    for response in range(0, len(responses)):
        if x["target" + str(response)] > 0:
            x["target"] = x["target" + str(response)]
            break;
    return x
```

Ввод [23]:
```
for response in range(0, len(responses)):
    model = logr_models[response]["model"]
    columns_opt = logr_models[response]["columns"]
    x = pd.DataFrame(data_test, columns=columns_opt)
    data_test["target" + str(response)] = model.predict(x)
```

Ввод [26]:
```
data_test = data_test.apply(logr_hierarchy, axis=1,
                            result_type="expand")
print (data_test.head())
```

```
                0         1         2         3         4         5         6  \
34624   0.895389  1.496539 -0.012580  0.134079  0.611857 -0.169414 -1.159587
49458  -0.160284  0.266282  1.197031 -0.329378  0.611857 -0.169414 -1.159587
8370   -0.583924 -0.463978 -0.391435 -0.427263  0.611857 -0.169414 -1.159587
23368   2.071710  0.516280 -0.921336  2.163699  0.611857 -0.169414  0.862391
56207   1.131202  1.739959  0.062943  0.231964 -1.634368 -0.169414  0.862391

                7         8         9  ...       114       115       116  \
34624   1.101046 -1.156735  1.130555  ... -0.623305 -0.216001 -0.128866
49458   1.101046 -1.156735  1.130555  ... -0.623305 -0.216001 -0.128866
8370    1.101046 -1.156735  1.130555  ...  1.604350 -0.216001 -0.128866
23368  -1.013721  0.866277 -0.928723  ... -0.623305 -0.216001 -0.128866
56207  -1.013721  0.867624 -0.928723  ... -0.623305 -0.216001 -0.128866

             117  Response  target0  target1  target2  target3  target
34624  -0.142142       7.0      0.0      0.0      7.0      4.0     7.0
49458  -0.142142       2.0      0.0      2.0      1.0      3.0     2.0
8370   -0.142142       8.0      8.0      0.0      1.0      4.0     8.0
23368  -0.142142       5.0      0.0      5.0      7.0      3.0     5.0
56207  -0.142142       2.0      0.0      0.0      7.0      4.0     7.0

[5 rows x 124 columns]
```

Кластеризация дает 0.192, kNN(100) - 0.3, простая лог. регрессия - 0.512

Ввод [29]:
```
print ("Логистическая регрессия, 4 уровня:",
       round(cohen_kappa_score(data_test["target"],
                  data_test["Response"], weights="quadratic"), 3))
```

```
Логистическая регрессия, 4 уровня: 0.496
```

Матрица неточностей

Ввод [28]:
```
print (confusion_matrix(data_test["target"],
                        data_test["Response"]))
```

```
[[ 457  331   38   23  197  421  132  128]
 [ 135  236    7    0   72   58   12    8]
 [  31   44   69   36   86  165   13   10]
 [  81   54   52  164   23  308   35   68]
 [  76  156   13    2  208   80   26   16]
 [  19   18    5   15   19  109   16   36]
 [ 252  271   16   11  350  788  858  511]
 [ 204  167    8   29  131  358  487 3128]]
```

МЕТОД ОПОРНЫХ ВЕКТОРОВ (SUPPORT-VECTOR MACHINE)

Постановка задачи

Загрузим данные, приведем их к числовым, заполним пропуски, нормализуем данные и оптимизируем память.

Разделим выборку на обучающую/проверочную в соотношении 80/20.

Построим модель опорных векторов (SVM) для наиболее оптимального разделения параметров на классы, используем несколько реализаций: линейную (LinearSVC) и через градиентный бустинг (SGDClassifier).

Проведем предсказание и проверим качество через каппа-метрику.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [11]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix
from sklearn.svm import LinearSVC
from sklearn.linear_model import SGDClassifier
from sklearn import preprocessing
```

Загрузка данных

Ввод [12]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Предобработка данных

Ввод [13]:
```
data["Product_Info_2_1"] = data["Product_Info_2"].str.slice(0, 1)
data["Product_Info_2_2"] = pd.to_numeric(data["Product_Info_2"].str.slice(1, 2))
data.drop(labels=["Product_Info_2"], axis=1, inplace=True)
for l in data["Product_Info_2_1"].unique():
    data["Product_Info_2_1" + l] = data["Product_Info_2_1"].isin([l]).astype("int8")
data.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
data.fillna(value=-1, inplace=True)
```

Набор столбцов для расчета

Ввод [14]:
```
columns_groups = ["Insurance_History", "InsuredInfo", "Medical_Keyword",
                  "Family_Hist", "Medical_History", "Product_Info"]
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
for cg in columns_groups:
    columns.extend(data.columns[data.columns.str.startswith(cg)])
print (columns)
```

```
['Wt', 'Ht', 'Ins_Age', 'BMI', 'Insurance_History_1', 'Insurance_History_2', 'Insurance_History_3', 'Insurance_Histor
y_4', 'Insurance_History_5', 'Insurance_History_7', 'Insurance_History_8', 'Insurance_History_9', 'Medical_Keyword_1
', 'Medical_Keyword_2', 'Medical_Keyword_3', 'Medical_Keyword_4', 'Medical_Keyword_5', 'Medical_Keyword_6', 'Medical_
Keyword_7', 'Medical_Keyword_8', 'Medical_Keyword_9', 'Medical_Keyword_10', 'Medical_Keyword_11', 'Medical_Keyword_12
', 'Medical_Keyword_13', 'Medical_Keyword_14', 'Medical_Keyword_15', 'Medical_Keyword_16', 'Medical_Keyword_17', 'Med
ical_Keyword_18', 'Medical_Keyword_19', 'Medical_Keyword_20', 'Medical_Keyword_21', 'Medical_Keyword_22', 'Medical_Ke
yword_23', 'Medical_Keyword_24', 'Medical_Keyword_25', 'Medical_Keyword_26', 'Medical_Keyword_27', 'Medical_Keyword_2
8', 'Medical_Keyword_29', 'Medical_Keyword_30', 'Medical_Keyword_31', 'Medical_Keyword_32', 'Medical_Keyword_33', 'Me
dical_Keyword_34', 'Medical_Keyword_35', 'Medical_Keyword_36', 'Medical_Keyword_37', 'Medical_Keyword_38', 'Medical_K
eyword_39', 'Medical_Keyword_40', 'Medical_Keyword_41', 'Medical_Keyword_42', 'Medical_Keyword_43', 'Medical_Keyword_
44', 'Medical_Keyword_45', 'Medical_Keyword_46', 'Medical_Keyword_47', 'Medical_Keyword_48', 'Medical_History_1', 'Me
dical_History_2', 'Medical_History_3', 'Medical_History_4', 'Medical_History_5', 'Medical_History_6', 'Medical_Histor
y_7', 'Medical_History_8', 'Medical_History_9', 'Medical_History_10', 'Medical_History_11', 'Medical_History_12', 'Me
dical_History_13', 'Medical_History_14', 'Medical_History_15', 'Medical_History_16', 'Medical_History_17', 'Medical_H
istory_18', 'Medical_History_19', 'Medical_History_20', 'Medical_History_21', 'Medical_History_22', 'Medical_History_
23', 'Medical_History_24', 'Medical_History_25', 'Medical_History_26', 'Medical_History_27', 'Medical_History_28', 'M
edical_History_29', 'Medical_History_30', 'Medical_History_31', 'Medical_History_32', 'Medical_History_33', 'Medical_
History_34', 'Medical_History_35', 'Medical_History_36', 'Medical_History_37', 'Medical_History_38', 'Medical_History
_39', 'Medical_History_40', 'Medical_History_41', 'Product_Info_1', 'Product_Info_3', 'Product_Info_4', 'Product_Info
_5', 'Product_Info_6', 'Product_Info_7', 'Product_Info_2_2', 'Product_Info_2_1D', 'Product_Info_2_1A', 'Product_Info_
2_1B', 'Product_Info_2_1C', 'Product_Info_2_1B']
```

Нормализация данных

Ввод [15]:
```
scaler = preprocessing.StandardScaler()
data_transformed = pd.DataFrame(scaler.fit_transform(pd.DataFrame(data,
                                                    columns=columns)))
columns_transformed = data_transformed.columns
data_transformed["Response"] = data["Response"]
```

Оптимизация памяти

Ввод [16]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус',
          round(100 * (start_mem - end_mem) / start_mem, 1), '%)')
    return df
```

Ввод [17]:
```
data_transformed = reduce_mem_usage(data_transformed)
print (data_transformed.info())
```

```
Потребление памяти меньше на 45.59 Мб (минус 75.1 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 134 entries, 0 to Response
dtypes: float16(133), int8(1)
memory usage: 15.1 MB
None
```

Разделение данных

Преобразуем выборки в отдельные наборы данных

Ввод [18]:
```
data_train, data_test = train_test_split(data_transformed,
                                         test_size=0.2)
data_train = pd.DataFrame(data_train)
data_test = pd.DataFrame(data_test)
print (data_train.head())
```

```
              0         1         2         3         4         5         6  \
8482  -1.236328 -0.164551 -2.841797 -0.892090 -0.083679  0.441650 -0.149292
29512 -0.011101 -0.164551  0.312256  2.375000 -0.083679 -2.263672 -0.149292
29210 -0.028320 -0.164551  0.312256  0.560059 -0.083679  0.441650 -0.149292
12988 -0.976074 -0.164551  0.312256  1.467773 -0.083679  0.441650 -0.149292
15046 -0.852539 -0.164551 -2.841797 -0.892090 -0.083679  0.441650 -0.149292

              7         8         9  ...       124       125       126  \
8482   0.819336  0.269287 -0.630859  ... -0.142456 -0.240112  0.961914
29512  1.803711  1.003906  0.144409  ... -0.142456 -0.240112 -1.887695
29210  0.592285 -2.179688 -1.641602  ... -0.142456 -0.240112  0.961914
12988  0.895020  1.003906  1.130859  ... -0.142456 -0.240112  0.147827
15046  0.819336 -0.955078 -0.372559  ... -0.142456  4.164062  0.147827

            127       128       129       130       131       132  Response
8482  -1.133789 -1.332031 -0.623535  4.628906 -0.128906 -0.14209         8
29512 -0.666992  0.750977 -0.623535 -0.215942 -0.128906 -0.14209         1
29210 -0.200073  0.750977 -0.623535 -0.215942 -0.128906 -0.14209         8
12988  2.134766 -1.332031  1.604492 -0.215942 -0.128906 -0.14209         6
15046 -0.200073  0.750977 -0.623535 -0.215942 -0.128906 -0.14209         6

[5 rows x 134 columns]
```

SVM

Выбираем направления преобразований исходных данных, чтобы различные классы можно было разделить гиперплоскостью по значениям параметров.

Ввод [19]:
```
x = pd.DataFrame(data_train, columns=columns_transformed)
model_lin = LinearSVC(max_iter=10000)
model_lin.fit(x, data_train['Response'])
```

```
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\sklearn\svm\_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[19]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=10000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

Ввод [20]:
```
model_sgd = SGDClassifier()
model_sgd.fit(x, data_train['Response'])
```

Out[20]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

Предсказание данных и оценка модели

Ввод [21]:
```
x_test = pd.DataFrame(data_test, columns=columns_transformed)
data_test["target_lin"] = model_lin.predict(x_test)
data_test["target_sgd"] = model_sgd.predict(x_test)
```

Кластеризация дает 0.192, kNN(100) - 0.3, лог. регрессия - 0.512/0.496

Ввод [22]:
```
print ("SVM (линейный):",
       round(cohen_kappa_score(data_test["target_lin"],
                               data_test["Response"], weights='quadratic'), 3))
print ("SVM (градиент):",
       round(cohen_kappa_score(data_test["target_sgd"],
                               data_test["Response"], weights='quadratic'), 3))
```

```
SVM (линейный): 0.95
SVM (градиент): 0.914
```

Матрица неточностей

Ввод [23]:
```
print ("SVM (линейный)\n",
    confusion_matrix(data_test["target_lin"], data_test["Response"]))
print ("SVM (градиент)\n",
    confusion_matrix(data_test["target_sgd"], data_test["Response"]))
```

```
SVM (линейный)
 [[1244    0    0    0    0    0    0    0]
 [   0 1205   54   21   82   15    0    0]
 [   0    7    6    0    0    0    0    0]
 [   0   20   37  105    2   32    5    0]
 [   0   43   15    3  479  215  105    0]
 [   0   77   98  151  372 1515  614    0]
 [   0    0    0    3  148  403  907    0]
 [   0    0    0    0    0    0    0 3894]]
SVM (градиент)
 [[1237    0    0    0    0    0    0    0]
 [   7 1042   63   30  161   76   18    0]
 [   0   24   15   12    3    2    1    0]
 [   0   36   30   87    6   79   13    0]
 [   0   91   20   21  385  334  155    0]
 [   0  128   74  122  312 1169  593    0]
 [   0   31    8   11  216  520  851    0]
 [   0    0    0    0    0    0    0 3894]]
```

Часть 5
АНСАМБЛЕВЫЕ МОДЕЛИ

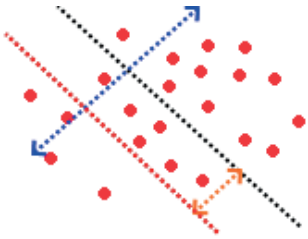
АНСАМБЛЕВЫЕ МОДЕЛИ

У каждой модели машинного обучения есть некоторый предел, до которого можно повышать точность, а дальше начинается переобучение и модель уже физически не может предсказать точнее, чем заложено в её природе. Эта природа определяется не только моделью, а связкой модели, то есть математический аппарат и данные, которые эта модель пытается преобразовать. Однако точность работы одной конкретной модели не является пределом при решении задач машинного обучения.

Когда с этим столкнулись первые исследователи они обратились к математической статистике и выяснили что, если взять несколько однородных моделей и среднее их предсказание, но при этом модели обучены на немного разных выборках. То получится, что ошибка среднего равна $\sqrt{n}$, где n это количество моделей. И ошибка выходит меньше за счёт того, что мы берём несколько одинаковых моделей.

Этот подход известен как ансамблирование. Так зачем нужно ансамблирование? В целом задачи машинного обучения сводятся к уменьшению смещения и статистической ошибки работы модели.

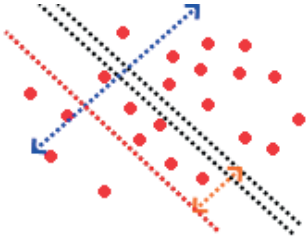
Предположим, что у нас есть двумерная прямая красного цвета, это наша идеальное состояние, чтобы модель машинного обучения ее предсказывала.



Эта прямая описана таким набором точек. И получилось так что наша выборка смещена (оранжевая линия) от этой модели, поскольку если построить характерную для выборки линию (черная линия), которая будет в центре выборки то получится линия смещенная от изначальной. Также имеется достаточно большой разброс (синяя линия).

Идеальная модель машинного обучения сместится максимально близко к нашей изначальной красной линии, которую мы хотим предсказать. И раз-

брос её, то есть ошибка предсказания тоже будет крайне мала. Желательно чтобы она была не меньше, чем смещение. То есть наша красная прямая попала в статистическую ошибку предсказаний. Потому что часто модель переобучается и сужается до такого канала:



Соответственно у нее малый разброс, она точно предсказывает значения. Но эти значения все смещены, и мы не можем им доверять. Получается если мы добавим некоторые тестовые данные вне этого разброса, то модель не будет корректно предсказывать эти данные.

Теперь разберем модель бэггинга. Она работает над уменьшением разброса. Когда мы берём очень много однородных моделей. Например, много решающих деревьев объединяем в случайный лес, модель получается однородная. Берём предсказания 100 деревьев считаем среднее и получим что ошибка в $\sqrt{100}$ раз меньше. Это получается за счет уменьшения разброса.

Если же мы работаем над уменьшением смещения, то это бустинг. Бустинг предполагает, что последовательно за счёт анализа данных, на которых мы промахиваемся, строим линии все ближе и ближе к нашим ошибкам. То есть мы усиливаем нашу модель, уменьшаемое её смещение относительно реальных данных, потому что наша выборка будет содержать истинные данные. Ну то есть будет содержать точные данные и нам важно найти эти точные данные и по ним построить модель.

Подход бустинга можно сравнить с учениками, решающими домашнее задание. Например, один ученик делает домашнее задание, но делает с ошибками. Он сверяет с домашним заданием другого ученика и смотрит свои ошибки и так у всего класса. В конце получается уже, верно, выполненное домашнее задание, которое мало смещено от идеального.

Выделяют также третий тип ансамбля – это ансамбль стекинга. Стекинг состоим в том, чтобы комбинировать разнородные модели. Например, у нас модель логистической регрессии, потом возьмём опорные векторы, затем возьмем еще случайный лес и так далее.

Получается, что стекинг это ансамбль ансамблей, либо ансамбль разнородных моделей. Бэггинг это ансамбль однородных моделей. Работает за счет уменьшения ошибки среднего, при этом уменьшая разброс и увеличивая точность работы. А стекинг за счет ошибки среднего сводит к минимуму смещение, зная, что разброс каждой конкретной модели уже небольшой.

То есть если мы с разных сторон от истинной прямой построим несколько разных вариантов, затем возьмём среднее то мы скорее всего попадем практически в нужную точку.

БУТСРЭП

Работа с ансамблями машинного обучения, с ансамблями бэггинга, в частности, базируется на идее, связанной с мультиплицированием данных. У нас есть исходный набор данных и проблема в том, что на всём наборе мы можем быть только одну модель.

Если мы разобьём набор на десять частей, чтобы обучить десять моделей, то потеряется репрезентативность наших моделей, получается если на одной десятой части будем обучать модель данных может просто не хватить. Выходит, что у нас может не быть столько данных, чтобы обучить большое количество независимых моделей для ансамблирования.

Рассмотрим метод бутсрэп. Он позволяет размножить и пере использовать данные чтобы обойти эту проблему. Есть несколько подходов бутсрэпа. Рассмотрим два из них: классический бутсрэп и «складной нож». Они отличаются тем, что бутсрэп выбирает элементы с возвращением, а складной нож без возвращения.

Рассмотрим пример, у нас есть двадцать элементов:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

И мы хотим на этой выборке обучить много моделей. Бутсрэп подразумевает, что мы выбираем столько элементов в подвыборку сколько у нас есть элементов в исходной выборке. То есть у нас есть двадцать, то выбираем тоже двадцать. Однако бутсрэп может быть и меньшего размера, например, из двадцати будем выбирать пятнадцать элементов, причем в случае бутсрэпа мы выбираем элемента с возвращением.

Суть метода: берём случайные элементы, например, 8, 15, 3, 8 и так выбираем двадцать элементов. Среди них могут быть повторения. Число повторений будет примерно равно 37 %. Соответственно уникальных данных примерно 63 %, это если мы делаем бутсрэп того же размера, что исходная выборка.

Зачем собственно нужен бутсрэп, поскольку мы выбираем элементы некоторым случайным образом, то получившееся подвыборка будет обладать практически теми же свойствами, то есть достаточно хорошо описывать нашу генеральную совокупность, все наши неизвестные случаи, которые мы пытаемся с помощью наших данных предсказать.

То есть у нас двадцать случаев и с помощью их хотим предсказать ещё двадцать миллионов. И чтобы эти двадцать миллионов лучше предсказать, из них выбираем двадцать получается первый усечённый набор данных, затем выбираем ещё раз и ещё раз, таким образом получаем какой-то большой набор усеченных наборов данных. Они, естественно, друг другом перекрываются, они дублируют друг друга. В этом ничего страшного нет. На этих подвыборках бутсрэп строим модели машинного обучения. Из-за того, что у нас выборки случайны мы считаем, что нас модели частично зависимы, то есть в каком-то смысле они независимы и они отражают отдельно взятую ситуацию от нашей исходной., то есть пытаемся из одного набора данных получить много ситуаций, пусть немного меньшего размера, но нам важно чтобы ситуаций было существенно больше.

Получается за счёт того, что уменьшаем размер данных на 37 %, мы существенно можем увеличить количество почти независимых ситуаций, которые описывают нашу неизвестную генеральную совокупность. За счет бутсрэпа мы можем построить много однородных моделей и дальше использовать бэггинг.

Метод складного ножа отличается от бутсрэпа тем, что мы используем элементы без повторений и обычно в складном ноже используется не все элементы исходной выборки. Поскольку если мы выберем без повторения все элементы случайным образом, то просто выберем наш набор данных.

Чтобы такого не происходило обычно в случае складного ножа выбирают примерно 60 % данных. Например, для нашего случая:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

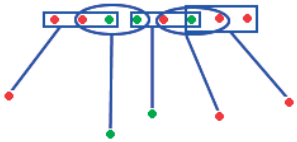
По сути метод складного ножа примерно идентичен бутсрэпу. Если мы будем выбирать 63 % исходных данных. Единственное складной нож нам дает гарантию, что будет ровно 63 % уникальных данных, которые мы выбрали, а в случае бутсрэп это число плавающее. Например: в какой-то подвыборке получим 30 % процентов не уникальных данных, в другой 25 %, в еще одной 40 %. Поэтому далее упоминая бутсрэп будем понимать, что используем метод складного ножа.

БЭГГИНГ

Предположим, что мы уже смогли с помощью бутсрэпа получить из исходного набора данных некоторый набор псевдонезависимых подвыборок. Теперь разберём бэггинг, расшифровывается как бутсрэп агрегация, то есть объединённая прокачка данных. То есть данные прокачали, размножили и теперь

можем их объединить и усреднить, но не сами данные ведь тогда мы получим исходный набор данных. Поэтому нам нужно объединить однородные модели, которые мы построим на этих данных.

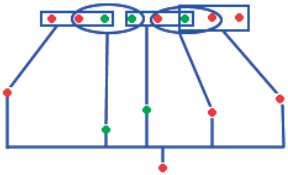
Предположим, что у нас есть восемь исходных значений. На основании этих восьми исходных значений построили пять моделей. У нас есть пять моделей, и они предсказывают класс объекта, который они чаще всего «видят» в нашей подвыборке:



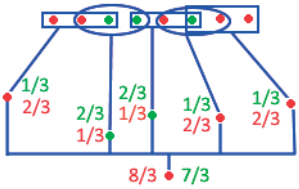
И теперь подаем на вход какой-то объект и сказать на какой класс он похож. Первая модель скажет красный круг, вторая зеленый круг, третья тоже зеленый круг, а четвертая и пятая скажут красный круг. Таким образом мы получили пять разных выходов у однородных моделей, которые обучены на подвыборках из наших данных.

Здесь могут быть любые однородные модели, которые были обучены на различных подвыборках нашей исходной обучающей выборки, то есть проводим бэггинг не на всем наборе данных, а только на обучающей выборке. Чтобы на тестовой уже проверить насколько хорошо получился бэггинг.

Теперь нам нужно агрегировать эти модели. Например, путем мажоритарного голосования. То есть каждой модели припишем один класс и получится красный кружок:

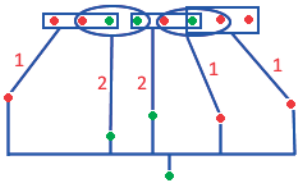


Также можно применить миноритарное голосование или «мягкое» голосование. В нем теперь есть не только самый популярный класс, а еще и вероятности некоторые вероятности нашей модели:



В данном случае мы получим 8/3 за красный и 7/3 за зеленый. В жестком голосовании было три голоса против двух, а в мягком голосовании получилось восемь против семи.

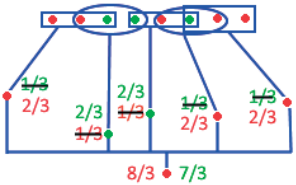
Кроме мягкого и мажоритарного голосования может быть взвешенное. Например, мы говорим, что эти модели имеют вес:



И теперь просуммировав веса можем сказать, что веса зеленых = 4, а веса красных = 3. Получается взвешенное голосование дало значение что это зеленый круг. Получается не всё так просто и за счёт весов в наших однородных моделях мы можем добиться того, что они будут по-разному себя вести в нужных нам ситуациях.

Эти методы голосования применяется не только в бэггинге, а также в стекинге, когда нужно несколько разнородных моделей сложить применятся тот же подход.

Кроме голосований также может применяться отбрасывание слабых вероятностей. Рассмотрим на нашем примере:



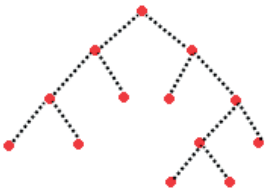
Считаем, что вероятность меньше 1/2 мы не рассматриваем, то есть считаем их несущественными. Получается убираем вероятности 1/3 и учитываем только 2/3. И далее они участвуют в голосовании. Выходит, что это четвертый подход к голосованию, который может применяется как в ансамблях бэггинга, так и в ансамблях стекинга.

СЛУЧАЙНЫЙ ЛЕС

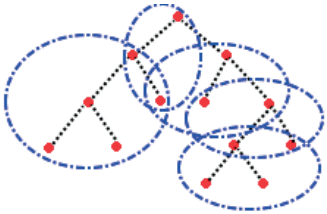
Случайный лес представляет собой ансамбль моделей, включает в себя все преимущества ансамбля бэггинга. Случайный лес собирается на основании деревьев принятия решений однородных моделей, которые берутся из псевдо-

независимых подвыборок обучающего набора данных. Однако, случайный лес содержит ещё одно ключевое отличие от бэггинга.

Оно содержится в его названии он случайный, то есть кроме того, что мы случайным образом выбираем данные, ещё и случайным образом выбираем часть наших гиперпараметров ансамбля моделей. Предположим, что у нас есть некоторое идеальное дерево принятия решений:

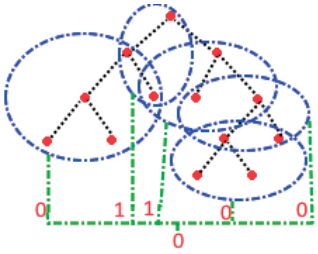


Оно включает в себя все параметры, которые у нас есть. В чём заключается случайный лес. Кроме того, чтобы на вход подаём, например, $\frac{N}{5}$, в пять раз меньше данных, то есть такая случайная выборка в пять раз меньшего размера данных которую даем на вход одному из деревьев. Это дерево не строится идеальным способом, поскольку строится по меньшему числу параметров и оно содержит меньше листьев:



Получается, что из некоторого идеального дерева, которое описывать все наши возможные значения данных, мы вырезаем части и говорим, что это наши урезанные однородные модели. Это привод к тому, что с некоторой каждая конкретная модель начинает предсказывать какой-то свой кусочек данных очень хорошо, поскольку она не знает про все остальные возможности как данных так параметров этих данных и она может обобщить свой небольшой участок данных.

Эти модели строятся по разному набору параметров, могут содержать разные разбиения этих параметров, потому что данные у них разные. И получается, что на одной и той же псевдослучайной подвыборке из нашей генеральной совокупности мы получаем разные её приближения. И это даёт основание предполагать, что разброс нашего ансамбля получается меньше. Например, делаем мажоритарное голосование:



За счёт голосования по всем деревьям получаем, что это объект 0. Объект, который нужно классифицировать подается на вход всех деревьев, каждое из деревьев принимает решение, потом это решение агрегируется. И на выходе мы получаем некоторое значение.

Подытожим. Случайный лес, кроме того, что он строит каждую отдельную модель на подвыборках, которые случайным образом получены из исходных данных, он строит ещё по некоторому урезанному случайному набору параметров. То есть рассматривает только часть параметров и пытается по этой части параметров наилучшим образом предсказать класс этих данных.

Затем на выходе, когда мы уже принимаем решение на основании усредненных предсказаний наших моделей и получаем финальный ответ этого ансамбля бэггинга, который называют случайный лес.

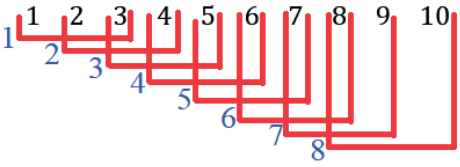
OUT-OF-BAG

Разберемся такое понятие как Out-of-Bag Error или ошибка выпадения данных. Несмотря на название это не является ошибкой и это понятие относятся исключительно к бэггингу. Поскольку мы выбираем не все данные из обучающих для того, чтобы построить каждую конкретную однородную модель, а немного меньше. У нас остается часть данных, на которых мы проверяем модель.

И мы можем эту особенность использовать чтобы дополнительно улучшить нашу модель. Рассмотрим на конкретном примере, есть такой набор данных:

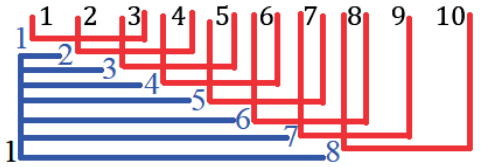
1 2 3 4 5 6 7 8 9 10

И на этих данных строим некоторый набор моделей:

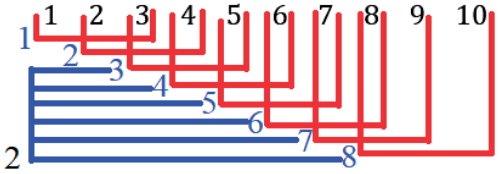


Модели пронумерованы синими цифрами, и каждая состоят из трех экземпляров данных из нашей выборки. Поскольку используется только три из десяти объектов, мы можем на остальных семи объектах проверить эту модель. Но так как у нас ансамбль бэггинга, то нам не нужно проверять эту модель в отдельности. Нам нужно проверить работу всего ансамбля. Но достоверно мы это проверить не можем, потому что часть моделей обучалась на тех данных на которых мы будем проверять, а часть нет.

Поэтому из всех моделей строится так называемый под-ансамбль, с теми же параметрами, например с равными весами голосования. И он проверяется на тех данных на которых модели в этом под-ансамбле не обучались. Разберем на нашем примере:



Получается, первый набор можно обучить на ансамбле из почти всех восьми моделей. Потому что первый объект данных у нас не вошёл ни в одну из моделей, кроме первой. То есть мы пересобрали наш под-ансамбль проверили на первом объекте, на нём этот ансамбль не обучался. И благодаря этому мы можем получить независимый результат. Второй объект не попал в модели с третьей по восьмую:



И так далее строим для всех объектов. Как видно в наборе получается несколько моделей минимум четыре модели. Иногда этот показатель доходит до 2/3. В результате выходит репрезентативная выборка однородных моделей, которые не обучались на определенных срезах данных. И на этих моделях можно проверить точность нашего ансамбля и соответственно проверить гиперпараметры ансамбля.

В этом смысле дополнительно разбивать наш обучающий набор на обучающий и проверочный будет избыточно, потому что мы можем проверить точность работы ансамбля бэггинга непосредственно при работе этого ансамбля.

Это и называется Out-of-Bag Error, которая позволяет при обучении сразу и проверить ансамбль на независимых данных. Однако если мы исполь-

зуем уже ансамбль стекинга, то есть у нас не только, например, случайный лес, а есть еще градиентный бустинг, логистическая регрессия и другие. И нам важно содержать их в одних условиях. Когда у нас разнородные модели машинного обучения объединяются в ансамбль стекинга, то здесь уже обучаем каждую конкретную модель на обучающей выборке, чтобы у них были равные условия. С тем чтобы проверить на проверочной выборке сам ансамбль стекинга.

Поскольку не у всех ансамблей есть это свойство Out-of-Bag Error, поэтому мы не сможем использовать его для всех. Однако если у нас единственная модель случайные лес и, например, сверхслучайный лес, соответственно можем обучать их на всей обучающей выборке без разделения на обучающие и проверочные части и используя Out-of-Bag Error можем проверить точность этой модели.

СВЕРХСЛУЧАЙНЫЕ ДЕРЕВЬЯ

На большой выборке из сильно разреженных данных, то есть разных данных с разными характеристиками и небольшого количества классов случайный лес может показывать невысокую обобщающую силу. Это происходит даже несмотря на то, что мы делаем случайную подвыборку из исходных данных, каждый раз мы разбиваем значение в листьях дерева по наименьшей энтропии и по факту даже для случайных подвыборок получаем примерно одинаковое переобучение.

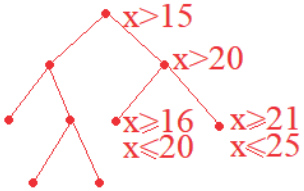
Выходит, что модель случайного леса дает хороший результат на обучающей выборке, однако на тестовой выборке она даёт результат существенно хуже, потому что каждый раз подгоняется под непонятные и разбросанные данные. То есть, когда много выбросов и данные сильно разрежены в пространстве признаков. Соответственно выходит, что случайный лес как-то подгоняет модель, вместо того чтобы отбросить часть данных как несущественные.

Добавим отбрасывание данных, то есть некоторый стохастический или случайный процесс в случайный лес. И получим сверхслучайные деревья или extremely randomized trees. В чем заключается подход сверхслучайных деревьев?

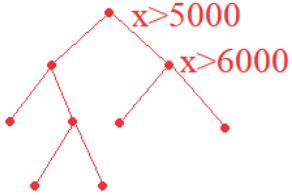
Сначала вспомним про решающие деревья. У нас есть некоторое дерево принятия решений. То есть мы смотрим на объект данных смотрим его признак и в зависимости от значения признака или диапазона значений признака мы адресуем его в ту или иную ветку дерева и таким образом доходим до конца.

При разбиении, как и по какому правилу нам оценивать признак в каждом конкретном узле? В случае случайного леса разбивается по минимуму энтропии. Получается нам важно выделить максимум информации из всего набора признаков всех объектов, чтобы вышло максимально разделить эти объекты на две группы с максимальным добавлением информации. Следова-

тельно, сделать объекты на следующем уровне более однородными, то есть должны принять решение по какому значению параметра объекта, чтобы упорядочить нашу выборку. Пример:



Подход сверхслучайных деревьев, как и в случае случайного леса представляет собой некоторое разбиение исходных данных:

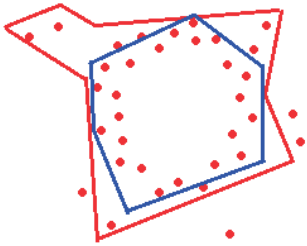


В случае сверхслучайных деревьев мы выбираем, например, пять значений из ряда значений 5000 < x < 6000. Сверхслучайные деревья похожи на случайный лес (используется случайное подмножество возможных признаков), однако пороговые значения выбираются случайно для каждого возможного признака (вместо поиска наиболее оптимальных порогов в случае случайных деревьев; кстати, такое отличие сверхслучайных деревьев добавляет в них больше «случайности» по сравнению с случайными деревьями).

Таким образом выстраивается деревья сверх случайного леса, которые могут объединятся в ансамбль бэггинга. И получаем, что за счёт случайного отброса, может быть, даже важных значений, мы фильтруем наши данные. И тут важно вспомнить, что модель дерева принятия решений является обобщением метода ближайших соседей, имеющий недостатки: выбросы данных сильно влияют на результат и вторая проблема, если имеется много однотипных объектов будем вынуждены вычислять значения по всем однотипным объектам вместо того, чтобы взять только половину.

Так вот сверхслучайный лес решает обе эти проблемы. Если у нас много объектов, то мы автоматически просто отсечём их. Если попадается явный выброс, мы можем случайным образом его не выбрать. Вероятность выбора выброса, в большинстве деревьев, низка. Получается сверхслучайный лес не обобщает случайные выбросы и соответственно хорошо работает, когда у нас много разреженных данных. Сравним случайны лес и сверхслучайный лес на примере:

Случайный лес (красная линяя) выдаст многомерный многоугольник, который будет пытаться охватить все, а сверхслучайный деревья (синяя линия) не сможет этого сделать, из-за того, что отбросят половину шумов. И в итоге они сведутся к многоугольнику, который наиболее характерно описывает наше пространство признаков:



Выходит, что когда у нас сильно разрежены данные сверхслучайные деревья будет работать лучше, если же данные компактные, то случайный лес отличный метод. Также можно использовать ансамбль стекинга сверхслучайных деревьев и случайного леса, на тот случай если не понятно данные компактны или разрежены.

В реальных задачах данные чаще разрежены, очень редко бывают компактными. Поэтому при принятии решения какую модель использовать используем обе модели и объединяем их в ансамбль стекинга. И помним, что они однородны. Поэтому в ансамбле стекинга лучше их использовать с весами 1/2.

АДАПТИВНЫЙ БУСТИНГ

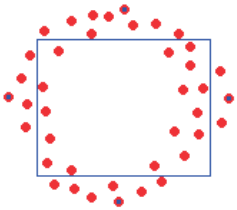
Кроме ансамбля бэггинга или подхода, когда мы берём много однородных моделей, усредняем их предсказания и считаем, что голос толпы у нас будет справедлив. То есть, когда много экспертов проголосовали мы выбираем наиболее вероятное голосование и скорее всего это действительно будет истина. Существует также подход, который мы используем буквально с самого рождения. Например, навык шитья, в начале нам даже сложно пришить пуго-

вицу, но с каждой попыткой будет все проще и проще и далее уже можно будет переходить на более сложные элементы шитья.

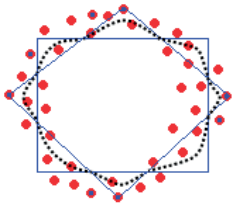
Этот подход в машинном обучении называется бустинг. Он заключается в том, что последовательно мы продвигаемся к некоторому мастерству. Мы люди так учимся и логично научить машины так обучаться. Рассмотрим на конкретном примере одну из вариаций этого подхода – адаптивный бустинг:

Модель адаптивного бустинга оперирует ансамблям, то есть это ансамблевая модель. Мы предполагаем, что нас у нас этот ансамбль уже построен, но он также может строится в процессе. Важно, что здесь модели строятся одинаковые, то есть если у нас дерево принятия решений, то каждый раз строим дерева принятия решений.

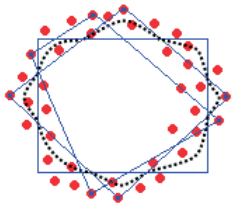
Однако вся суть адаптивного бустинга, в том на каких данных мы строим дерево принятия решений. Если, например, выбираем бутсрэп выборку, делая некоторую подвыборку из нашей обучающей выборки, то нам важно понять на каких данных нужно обучать модель адаптивного бустинга. Предположим, мы построили первую модель, и она обобщила некоторый квадрат из нашей окружности:



Дальше смотрим на те данные, которые больше всего выпали из нашей модели. Получилось примерно четыре точки, которые дальше всего стоят. Мы эти данные выбираем, для того чтобы на них построить следующую модель. Правда идет выбор не самих данных, а обновление весовых коэффициентов этих данных, то есть эти данные с большей вероятностью попадут в следующую бутсрэп выборку, нашего обучающего набора, и мы по ним построим следующие модель или обновим веса. Соответственно следующую модель строим примерно такую:



Если изобразить усреднение, то оно уже хорошо напоминает окружность, то есть модель адаптивного бусинка позволила приблизится ближе к решению. Повторим итерацию, выделим еще четыре точки мимо которых промахнулись и построим еще одну модель:



То есть мы построим следующие однородные модели на тех же данных, на которых строятся остальные модели, однако мы выбираем из этих данных с большими весами, те по которым промахнулись. Если последовательно две модели промахнулись по данным, то эти данные с большей вероятностью попадут в подвыборку для следующих моделей.

Построив еще некоторое количество моделей, дойдем до такого сглаженного многоугольника, который будет воспроизводить нашу окружность. В итоге адаптивный бустинг заключается в том, что:

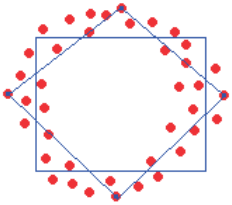
1. Делаем подвыборку данных.
2. Обучаем модель.
3. Проверяем как работает наша модель и назначаем максимальные веса объектам, для их участия в построении модели.
4. Вычисляем ошибку
5. Обновляем значения весов.

И с обновленными весами возвращаемся в начало. Уже условно через сто итераций сойдемся с моделью, которая будет иметь обобщающую силу. Дальше мы просто уже на тестовом наборе наших данных проверим, чтобы модель сохраняла обобщающую силу, то есть стандартными методами сделаем оптимизацию параметров и проведём контроль переобучения модели.

LOGITBOOST, BROWNBOOST И L2BOOST

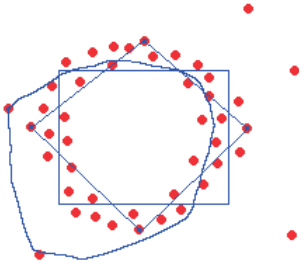
Адаптивный бустинг является отличным выбором если у нас есть некоторые компактные данные. Но на хороших данных любой алгоритм работает хорошо и нам важно понять, как можно модифицировать адаптивный бустинг, чтобы даже на разреженных данных или сильно зашумленных данных он давал всё ещё хорошие значение.

Возьмем пример из прошлой части:



Последовательно выбираем четыре точки, по ним строим квадраты, и затем они объединяются в нужную нам фигуру. То есть проводим некоторые обобщения, каждая модель имеет собственный вес, и мы обновляем веса данных.

Однако если будет какое-то количество выбросов, то модель не будет сходиться и будет большая ошибка. Но самое главное, у нее не будет обещающей силы. В какой-то момент остановки обучения получим следующую модель:

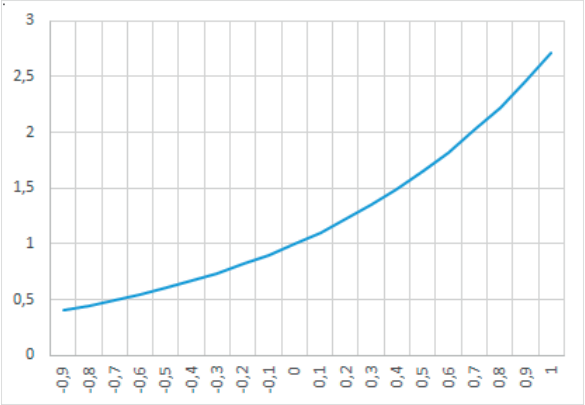


Чтобы побороть эту проблему используют более строгую функцию пенализации плохих данных. Адаптивный бустинг базируется на том, что на следующей итерации мы умножаем веса на некоторую отрицательную экспоненту:

$$w_{i+1} = w_i * e^{-\lambda|y-y_i|x_i}.$$

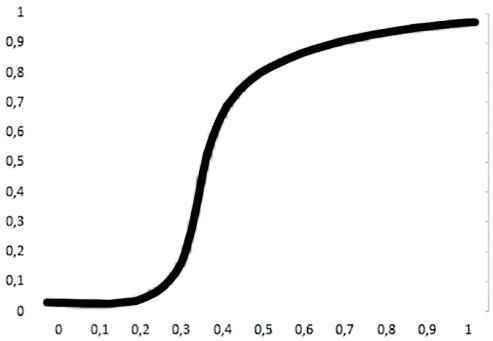
Нам нужно увеличить веса тех данных, которые у нас предсказаны плохо, чтобы они попали в выборку следующей модели и при этом уменьшить

веса тех данных, которые у нас предсказаны хорошо. Это делается за счёт экспоненциальной функции:



Поскольку значение предсказание обычно от нуля до единицы или от минус единицы до единицы, то весовое значение меняется примерно в 10 раз. Но если мы, верно, предсказываем данные нам не так существенно насколько, верно, мы предсказываем. Например, у нас выходит больше 0,9 верных предсказаний при бинарной классификации, и мы можем дальше не двигаться в эту сторону, потому что уже достигли нужной точности. И наоборот если неверно, то нам не так критично насколько неверно, нам важно чтобы неверные предсказания быстрее попали в верную область.

Вот для того, чтобы решить эту проблему, а также справится с проблемой переобучения на неверных данных. Вместо экспоненты используют логистическую функцию:



Она имеет более крутой изгиб, чем экспонента. Изменяется от нуля до единицы. Этот рабочий промежуток существенно круче чем у экспоненты.

У логистической функции на отрезке от нуля до единицы, происходит изменение значений функции в сто раз. Получается веса обновляются достаточно консервативно в случае адаптивного бустинга, и мы пытаемся исправить этот подход используя LogitBoost (логистический бустинг). То есть при обновлении весов используем не экспоненту, а логистическую функцию.

И получаем, что неверно предсказанные веса быстрее уйдут в отрицание, то есть выберем их быстрее, а верно предсказанные быстрее пенализируем и меньше начнем учитывать. И соответственно совсем неверные предсказания, то они всё время будут болтаться в хвосте, и они с трудом смогут выйти в верные предсказания.

Поэтому с переобучением данная модель работает также не очень хорошо. Она лучше работает с обучением, то есть мы быстрее можем выделить не совсем характерны данные и обучить ансамбль на них. Однако на шумовых данных эта модель все также пытается обучиться. То есть через несколько итераций остаются только шумовые данные, которые мы не можем нормально предсказать.

И вот для того, чтобы решить проблему переобучения и не обучения на выбросах применяют BrownBoost. В чём заключается этот подход? В нем также используется логистическая функция, однако мы даём тем данным, которые, например, за десять итераций, так и не смогли нормально эти данные обобщить, значит класс наших моделей не может с ними никак работать, и они выбрасываются из рассмотрения.

Получается, что мы совсем плохие данные мы не включаем в следующую модель. Мы можем в них ошибаться, зато все остальные данные предскажем лучше. В этом плане BrownBoost, является среди алгоритмов адаптивного бустинга наиболее продвинутым и позволяет добиться большой точности, самой большой точности из всего семейства и также быстрее обучается за счет использования логистической функции.

При рассмотрении линейной регрессии также иногда упоминается подход адаптивного бустинга – L2Boost. Он заключается в том, что используем L2 регуляризацию весов. L2 регуляризация заключается в:

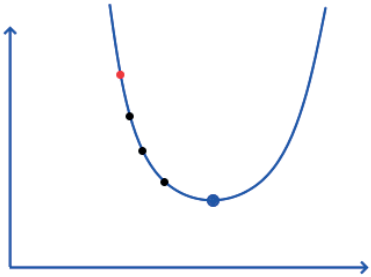
$$\sum \beta w_i^2.$$

Минимизации квадрата весов. Проблема поиска β решается только за счёт жадного поиска в линейной регрессии. Для того чтобы найти ее адаптивными методами и применяется L2Boost. Когда мы выбираем какое-то начальное значение и потом пытаемся понять какие данные не учли перед начальным значением и затем двигаться в сторону учёта лучшего предсказания данных используя уже модель линейной регрессии и L2 регуляризацию для нее.

ГРАДИЕНТНЫЙ СПУСК

Градиентный спуск – это математический аппарат, который позволяет найти экстремум минимум или максимум, в основном локальный, какой-то неизвестной или известной функции. Градиентный спуск, например, применяется в методе наименьших квадратов. Там есть функция ошибки собственно сумма квадратов отклонений предсказанных значений и реальных значений и, если вычислить сумму квадратов этих отклонений, мы смещаемся в сторону обратную предсказанной ошибки и получить точные значения.

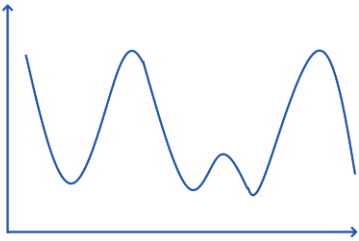
Когда известна точная функция, которая описывает некоторый «Ландшафт» ошибки, то есть в случае квадратов у нас есть некоторая многомерная парабола:



Предсказав какое-то значение в красной точке, знаем, что минимум ошибки будет находиться в вершине параболы, это в случае наименьших квадратов. Мы смещаемся в эту сторону итерационно, например по черным точкам и доходим минимума, там и останавливаемся.

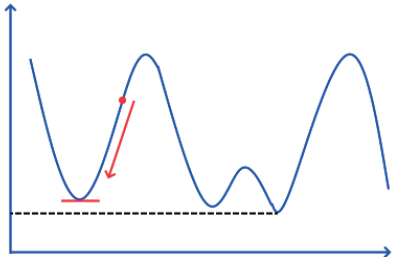
Но поскольку в случае среднеквадратичной ошибки функция ошибки определена явно для всех наших переменных во всех точках. То можем её сразу использовать, найти производную и вычислить минимум этой функции, приравняв производную к нулю. Выходит, что мы аналитически нашли этот минимум.

Однако в том случае если аналитическое вычисление минимума затруднено, например у нас система из миллиона на миллион параметров или вид функции, описывающей ландшафт неизвестен. Например, такая функция ошибки:



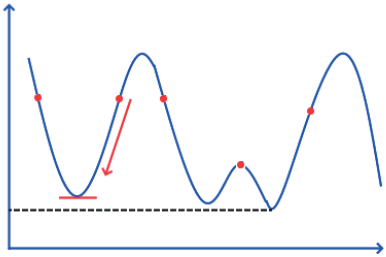
Данная функция одномерная, но она может быть и многомерной, в зависимости от числа гиперпараметров, и в таких функциях нам нужно понять в какую сторону двигаться. Градиентный спуск даёт нам примерно однозначный ответ в какую сторону нужно двигаться, чтобы дойти до какого-то подобия минимума, хотя бы локального минимума наших ошибок.

Поиск глобального минимума в общем случае – это очень тяжелая вычислительная задача. Всё же на текущий момент, есть подходы позволяющие обойти решение этой задачи в лоб. Например, берем предыдущую функцию:



Мы находимся в красной точке, начинаем двигаться в эту сторону доходим до некоторого минимума. Но глобальный минимум находится на пунктирной линии. Хорошая новость что глобальный минимум не сильно отличается от локального это довольно обычная ситуация.

Для того чтобы градиентный спуск работал лучше делается некоторый посев:

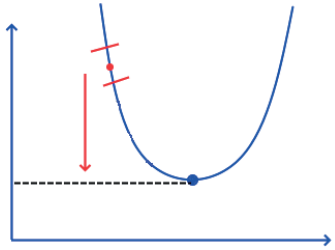


То есть выбирается, например, такой набор точек через равные промежутки. Затем выбрав по сетке некоторые значения пытаемся через методы дихотомии уменьшить этот отрезок. Это поможет нам двигаться в сторону минимума нашей функции ошибки.

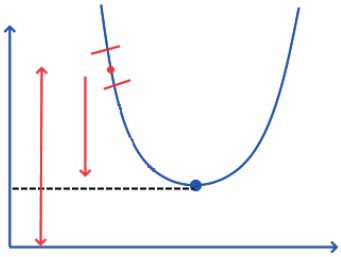
В простом случае функция ошибки – это разница между предсказанными и реальными значениями, соответственно если мы знаем где больше всего ошиблись. То есть у нас ансамбль модели и знаем какая из моделей ошиблась больше всего, то мы можем исправить эту модель в лучшую сторону. Если у нас есть предсказанные вероятности классов по классификации,

зная в каких местах их вероятности сильнее ошибаются, то исправляем в сторону обратную исправлению ошибки.

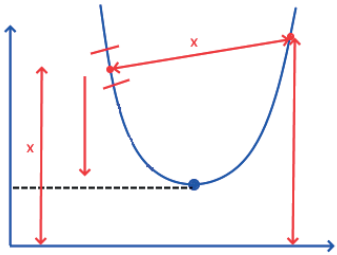
Важно также иметь в виду, что нельзя просто так взять и уменьшить ошибку до нуля. Это возможно только если знаем точное решение и можем точно перескочить на нужное значение. Когда используем градиентный спуск мы никогда не знаем на какой шаг нужно шагнуть, поскольку каждый раз шагая мы двигаемся вдоль функции. Но знаем мы только одно абсолютное значение, а ландшафт функции ошибки остается скрытым для нас. Также знаем, что в ту сторону ошибка меньше и хотим в ту сторону двигаться:



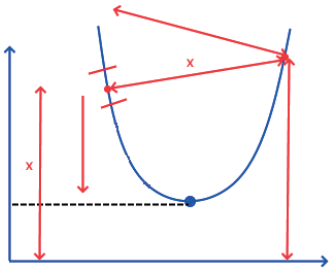
Получается, что двигаемся в эту сторону и знаем некоторые окрестности функции ошибки. И логично было бы сдвинуться на такое расстояние:



И кажется, что сейчас минимизируем ошибку, но если это расстояние отложить от текущей точки вдоль функции ошибки, то попадем примерно в эту точку:



И получилось, что ошибка стала, наоборот, немного больше. А если бы парабола была бы еще выше, то можно было бы вообще «вылететь» в другую сторону и этот процесс называется «раскачка». Мы хотели уменьшать ошибку, но на самом деле мы начинаем двигаться так и вылетаем за пределы этого локального минимума:



Чтобы этого не происходило обычно смещаемся не на величину ошибки, а на некоторую долю от этой ошибки, называемой LR – скорость обучения. Подбор скорости обучения это из наиболее замысловатых подборов гиперпараметров, потому что просто сетку наложить на него нельзя, поскольку это как бы гиперпараметр. Он определяет каким образом мы будем обучаться по текущим гиперпараметрам и уменьшать ошибку или даже изменять текущие гиперпараметры.

С какой скоростью нужно это делать чтобы процесс сошёлся хотя бы до локального минимума? Поскольку у нас максимальное значение единица, минимальное ноль, то диапазон изменения скорости обучения небольшой и можем жадным поиском проверить несколько значений. Для обычных задач классификации используют значения от 0,1 до 0,8.

Для нейронных сетей из-за более сложной функции ошибки, значения могут изменятся от $LR = e^{-10}...e^{-1}$. Такой маленький шаг обусловлен большой вероятностью вылета за пределы локального минимума.

Итак, градиентный спуск — это математический аппарат, который позволяет в наших предсказанных значениях понять в какую сторону нужно двигаться и примерно насколько большой шаг делать следующим. Этот шаг обычно умножается на скорость обучения. В случае нейронных сетей количество итераций может быть десятки тысяч, а в градиентном бустинге и ансамбле классификаций обычно достаточно несколько десятков итераций.

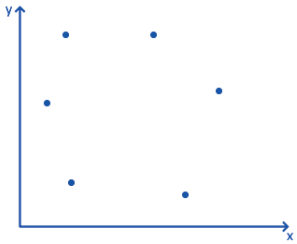
ГРАДИЕНТНЫЙ БУСТИНГ И XGBOOST

Градиентный бустинг является ещё одним направлением использования бусина, то есть мы всё также последовательно улучшаем модель, пытаемся её сделать идеальной за счёт последовательных приближений. Но в отличие от

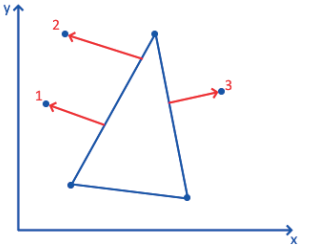
адаптивного бустинга, мы не обучаем несколько моделей на тех данных которые предсказали плохо. Поскольку не пытаемся лучше предсказать данные, а пытаемся исправить ошибки.

Например, мы стреляем в цель, то мы не будем стрелять наугад и брать усреднение. Если промахнулись стреляя вправо, то в следующий раз нужно взять левее. Мы пытаемся компенсировать ошибку, идём по градиенту ошибки, в том направлении, где ошибается модель и это направление мы предсказываем.

Рассмотрим на конкретном примере:

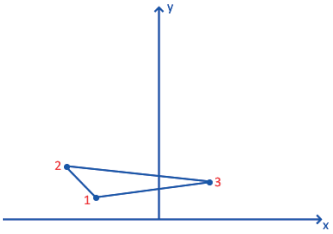


У нас есть шесть точек, и мы хотим их приблизить каким-то многогранником. Скажем, что у нас сначала есть треугольник, которым взяли выборку из этих точек:

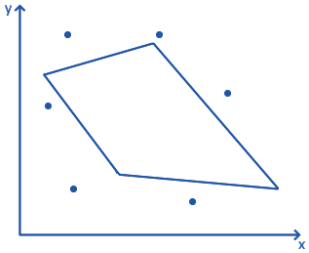


Этот треугольник промахнулся мимо трёх точек, направления ошибок представлены в виде красных стрелок. Когда используем градиентный бустинг мы обучаем модели последовательно не на самих данных, а на их ошибках, то есть разности значений между предсказанными и самими данными.

Следующая модель будет выглядеть примерно так:



Важно, что это точки ошибок, те значения разности. Они будут иметь такую же размерность как наши данные, по каждому параметру объекта данных вычисляем разницу и это набор этих разниц становится новым объектом. И далее объединяем с предыдущей моделью и получим примерно следующую модель:



Мы приблизились к попаданию во все данные, но все еще мимо. Такая у нас получилась вторая итерация. И далее также нужно вычислить ошибки второй модели, уже шесть точек, из них выбираем три. На них построим новую модель, она тоже каким-то образом подгонит фигуру под наши данные. Таким образом на какой-то итерации получим очень хорошую подгонку под исходные данные.

Одной из лучших реализаций градиентного бустинга является библиотека XGBoost или eXtreme Gradient Boosting, или же экстремальный градиентный бустинг. Библиотека содержит не только классический градиентный бустинг, а еще и L2 регуляризацию. То есть мы не позволяем некоторым моделям иметь слишком высокие веса.

XGBoost отлично борется с статистическими выбросами. Еще одной важной способностью является оптимизация решающих деревьев, то есть удаление дублирующих данных. Все эти достоинства позволяют использовать XGBoost в промышленных объемах.

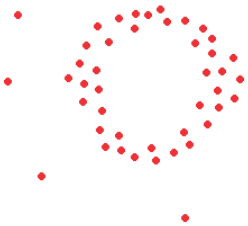
СТОХАСТИЧЕСКИЙ ГРАДИЕНТНЫЙ БУСТИНГ

Стохастический градиентный бустинг является наилучшим способом «прокачки» модели классификации. Лучшая комбинация достигается за счет прокачки решающих деревьев и максимального уточнения их предсказаний, при этом избегая переобучения деревьев. Нам важно чтобы задачи классификации решалась быстро, то есть мы дали на вход какой-то объект, нам быстро дали ответ и при этом мы в достаточной степени точно смогли обобщить неизвестные ранее данные.

В задачах бэггинга проблема переобучение решалась тем, что мы создаём много моделей на основе поднаборов объектов и поднабор параметров

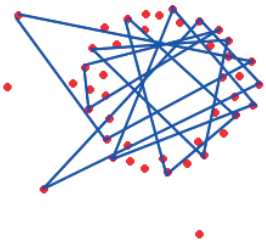
этих объектов. И получается, что наиболее характерные имеют большее влияние, а менее характерные выпадают из рассмотрения. Выходит, что с одной стороны решаем проблему большого количества типичных объектов, а с другой стороны решаем проблему статистического шума, то есть переобучение модели на нехарактерных выбросах.

Стохастический градиентный бустинг является совмещением идей градиентного бустинга и бэггинга. Рассмотрим пример:



В стохастическом градиентном бустинге не нужно брать все данные. Рассматриваем случайный поднабор данных и каждый раз ищем ошибку. Статистические выбросы будут попадать реже в наше поле зрения, поскольку подбираем маленький поднабор данных.

Из-за этого стохастический градиентный бустинг работает дольше, потому что ему до пяти раз больше нужно обучиться, чтобы достичь той же точности, как, например, у обычного градиентного бустинга. Однако за счет отброса переобученности, получаем уже более сбалансированную модель. На нашем примере это можно отобразить, используя треугольники:



За счет того, что мы берем меньше данных, в каждом треугольнике шанс на одновременное попадание двух выбросов мал, поэтому они получаются менее зашумлённые. Также становится видно, что треугольники начинают кучковаться в форме, которую мы пытаемся предсказать. Еще поскольку мы делаем этот процесс чаще, то наиболее характерные данные начинают попадать в нашу выборку лучше.

Стохастический градиентный бустинг используется как рабочая модель в большинстве задач, требующих числовых данных. Если нет категориальных данных, то стохастический градиентный бустинг даёт максимальную точность, которую можно ожидать от модели машинного обучения на задачах классификации. При категориальных данных стохастический градиентный бустинг работает хуже, есть модели, которые могут его обогнать.

ЧАСТЬ ПРАКТИЧЕСКИХ НАВЫКОВ К 5

РЕШАЮЩИЕ ДЕРЕВЬЯ

Постановка задачи

Загрузим данные, приведем их к числовым, заполним пропуски, нормализуем данные и оптимизируем память.

Разделим выборку на обучающую/проверочную в соотношении 80/20.

Построим несколько моделей дерева решений, найдем оптимальную через перекрестную валидацию (CV).

Проведем предсказание и проверим качество через каппа-метрику.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix, make_scorer
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn import preprocessing
from IPython.display import SVG, display
from graphviz import Source
import os
os.environ["PATH"] += (os.pathsep +
                       'C:/Program Files (x86)/Graphviz2.38/bin/')
```

Загрузка данных

Ввод [2]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Предобработка данных

Ввод [3]:
```
data["Product_Info_2_1"] = data["Product_Info_2"].str.slice(0, 1)
data["Product_Info_2_2"] = pd.to_numeric(data["Product_Info_2"].str.slice(1, 2))
data.drop(labels=["Product_Info_2"], axis=1, inplace=True)
for l in data["Product_Info_2_1"].unique():
    data["Product_Info_2_1" + l] = data["Product_Info_2_1"].isin([l]).astype("int8")
data.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
data.fillna(value=-1, inplace=True)
```

Набор столбцов для расчета

"Облегченный" вариант для визуализации дерева

Ввод [4]:
```
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
```

Нормализация данных

Ввод [5]:
```
scaler = preprocessing.StandardScaler()
data_transformed = pd.DataFrame(scaler.fit_transform(pd.DataFrame(data,
                                                                  columns=columns)))
columns_transformed = data_transformed.columns
data_transformed["Response"] = data["Response"]
```

Оптимизация памяти

Ввод [6]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                             / start_mem, 1), '%)')
    return df
```

Ввод [7]:
```
data_transformed = reduce_mem_usage(data_transformed)
print (data_transformed.info())
```

```
Потребление памяти меньше на 1.76 Мб (минус 77.5 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Data columns (total 5 columns):
0           59381 non-null float16
1           59381 non-null float16
2           59381 non-null float16
3           59381 non-null float16
Response    59381 non-null int8
dtypes: float16(4), int8(1)
memory usage: 522.0 KB
None
```

Разделение данных

Преобразуем выборки в отдельные наборы данных

Ввод [8]:
```
data_train, data_test = train_test_split(data_transformed,
                                         test_size=0.2)
data_train = pd.DataFrame(data_train)
data_test = pd.DataFrame(data_test)
print (data_train.head())
```

```
              0         1         2         3  Response
40140  0.778809  1.003906  0.970703  0.311279         2
24402  0.238281  0.514160 -0.467285  0.000043         7
57279 -0.161133  1.003906 -1.375977 -0.737793         6
27464 -1.594727 -0.955078 -1.073242 -1.542969         8
21574  0.191284  0.514160 -0.315918 -0.055450         8
```

Дерево решений

Минимальное число "одинаковых" значений для ветвления - 10

Ввод [9]:
```
x = pd.DataFrame(data_train, columns=columns_transformed)
model = DecisionTreeClassifier(random_state=0, min_samples_leaf=10)
model.fit(x, data_train["Response"])
```

Out[9]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=10, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

Визуализация дерева

Доступно несколько форматов вывода, большой SVG выводится в Jupyter Notebook сложнее, поэтому используем PNG.

В качестве названий параметров передаем исходный список.

Ввод [10]:
```
graph = Source(export_graphviz(model, out_file=None,
        feature_names=columns, filled=True,
        class_names=data_train["Response"].unique().astype("str")))
with open("tree.png", "wb") as f:
    f.write(graph.pipe(format="png"))
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.104915 to fit
```

"Дерево решений"

Влияние признаков

Выведем долю влияния признаков на конечный ответ в дерева

Ввод [11]:
```
print (model.feature_importances_)
```

```
[0.15368974 0.03832401 0.26152987 0.54645638]
```

Перекрестная проверка (CV)

Разбиваем обучающую выборку еще на k (часто 5) частей, на каждой части данных обучаем модель. Затем проверяем 1-ю, 2-ю, 3-ю, 4-ю части на 5; 1-ю, 2-ю, 3-ю, 5-ю части на 4 и т.д.

В итоге обучение пройдет весь набор данных, и каждая часть набора будет проверена на всех оставшихся (перекрестным образом).

Перекрестная проверка используется для оптимизации параметров исходной модели - решающего дерева в данном случае. Зададим несколько параметров для перебора и поиска самой точной модели.

Для проверки будем использовать каппа-метрику.

Ввод [12]:
```
tree_params = {
    'max_depth': range(10, 20),
    'max_features': range(1, round(len(columns_transformed))),
    'min_samples_leaf': range(20, 100)
}

tree_grid = GridSearchCV(model, tree_params, cv=5, n_jobs=2,
            verbose=True, scoring=make_scorer(cohen_kappa_score))
tree_grid.fit(x, data_train["Response"])
```

```
Fitting 5 folds for each of 2400 candidates, totalling 12000 fits
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:   43.6s
[Parallel(n_jobs=2)]: Done 196 tasks      | elapsed:   60.0s
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed:  1.5min
[Parallel(n_jobs=2)]: Done 796 tasks      | elapsed:  2.3min
[Parallel(n_jobs=2)]: Done 1246 tasks      | elapsed:  3.6min
[Parallel(n_jobs=2)]: Done 1796 tasks      | elapsed:  4.8min
[Parallel(n_jobs=2)]: Done 2446 tasks      | elapsed:  6.6min
[Parallel(n_jobs=2)]: Done 3196 tasks      | elapsed:  8.2min
[Parallel(n_jobs=2)]: Done 4046 tasks      | elapsed: 10.4min
[Parallel(n_jobs=2)]: Done 4996 tasks      | elapsed: 12.9min
[Parallel(n_jobs=2)]: Done 6046 tasks      | elapsed: 15.5min
[Parallel(n_jobs=2)]: Done 7196 tasks      | elapsed: 18.4min
[Parallel(n_jobs=2)]: Done 8446 tasks      | elapsed: 21.3min
[Parallel(n_jobs=2)]: Done 9796 tasks      | elapsed: 24.4min
[Parallel(n_jobs=2)]: Done 11246 tasks      | elapsed: 27.7min
[Parallel(n_jobs=2)]: Done 12000 out of 12000 | elapsed: 29.7min finished
```

Out[12]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=10,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=0, splitter='best'),
             iid='deprecated', n_jobs=2,
             param_grid={'max_depth': range(10, 20),
                         'max_features': range(1, 4),
                         'min_samples_leaf': range(20, 100)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(cohen_kappa_score), verbose=True)
```

Выведем самые оптимальные параметры и построим итоговую модель

Ввод [14]:
```
print (tree_grid.best_params_)
model = DecisionTreeClassifier(random_state=17,
        min_samples_leaf=tree_grid.best_params_['min_samples_leaf'],
        max_features=tree_grid.best_params_['max_features'],
        max_depth=tree_grid.best_params_['max_depth'])
model.fit(x, data_train["Response"])
```

```
{'max_depth': 10, 'max_features': 3, 'min_samples_leaf': 97}
```

Out[14]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=10, max_features=3, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=97, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=17, splitter='best')
```

Предсказание данных и оценка модели

Ввод [15]:
```
x_test = pd.DataFrame(data_test, columns=columns_transformed)
data_test["target"] = model.predict(x_test)
```

Кластеризация дает 0.192, kNN(100) - 0.3, лог. регрессия - 0.512/0.496, SVM - 0.95

Ввод [17]:
```
print ("Решающее дерево:",
       round(cohen_kappa_score(data_test["target"],
                  data_test["Response"], weights="quadratic"), 3))
```

```
Решающее дерево: 0.314
```

Матрица неточностей

Ввод [19]:
```
print ("Решающее дерево:\n", confusion_matrix(data_test["target"],
                  data_test["Response"]))
```

```
Решающее дерево:
 [[ 176  129    6    2   58  118   73   42]
 [  89  206    8    1   87   54   26   27]
 [   0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]
 [  72  147   39    0  354  122    7    4]
 [ 245  279   60   46  277  727  401  179]
 [  62   65   10    8   49  144  191   76]
 [ 539  491   90  263  296 1059  920 3553]]
```

СЛУЧАЙНЫЙ ЛЕС

Постановка задачи

Загрузим данные, приведем их к числовым, заполним пропуски, нормализуем данные и оптимизируем память.

Разделите выборку на обучающую/проверочную в соотношении 80/20.

Построим параллельный ансамбль (бэггинг) решающих деревьев, используя случайный лес.

Проведем предсказание и проверим качество через каппа-метрику.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix, make_scorer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn import preprocessing
```

Загрузка данных

Ввод [2]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Предобработка данных

Ввод [3]:
```
data["Product_Info_2_1"] = data["Product_Info_2"].str.slice(0, 1)
data["Product_Info_2_2"] = pd.to_numeric(data["Product_Info_2"].str.slice(1, 2))
data.drop(labels=["Product_Info_2"], axis=1, inplace=True)
for l in data["Product_Info_2_1"].unique():
    data["Product_Info_2_1" + l] = data["Product_Info_2_1"].isin([l]).astype("int8")
data.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
data.fillna(value=-1, inplace=True)
```

Набор столбцов для расчета

Ввод [4]:
```
columns_groups = ["Insurance_History", "InsuredInfo", "Medical_Keyword",
                  "Family_Hist", "Medical_History", "Product_Info"]
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
for cg in columns_groups:
    columns.extend(data.columns[data.columns.str.startswith(cg)])
print (columns)
```

```
['Wt', 'Ht', 'Ins_Age', 'BMI', 'Insurance_History_1', 'Insurance_History_2', 'Insurance_History_3', 'Insurance_Histor
y_4', 'Insurance_History_5', 'Insurance_History_7', 'Insurance_History_8', 'Insurance_History_9', 'Medical_Keyword_1
', 'Medical_Keyword_2', 'Medical_Keyword_3', 'Medical_Keyword_4', 'Medical_Keyword_5', 'Medical_Keyword_6', 'Medical_
Keyword_7', 'Medical_Keyword_8', 'Medical_Keyword_9', 'Medical_Keyword_10', 'Medical_Keyword_11', 'Medical_Keyword_12
', 'Medical_Keyword_13', 'Medical_Keyword_14', 'Medical_Keyword_15', 'Medical_Keyword_16', 'Medical_Keyword_17', 'Med
ical_Keyword_18', 'Medical_Keyword_19', 'Medical_Keyword_20', 'Medical_Keyword_21', 'Medical_Keyword_22', 'Medical_Ke
yword_23', 'Medical_Keyword_24', 'Medical_Keyword_25', 'Medical_Keyword_26', 'Medical_Keyword_27', 'Medical_Keyword_2
8', 'Medical_Keyword_29', 'Medical_Keyword_30', 'Medical_Keyword_31', 'Medical_Keyword_32', 'Medical_Keyword_33', 'Me
dical_Keyword_34', 'Medical_Keyword_35', 'Medical_Keyword_36', 'Medical_Keyword_37', 'Medical_Keyword_38', 'Medical_K
eyword_39', 'Medical_Keyword_40', 'Medical_Keyword_41', 'Medical_Keyword_42', 'Medical_Keyword_43', 'Medical_Keyword_
44', 'Medical_Keyword_45', 'Medical_Keyword_46', 'Medical_Keyword_47', 'Medical_Keyword_48', 'Family_Hist_1', 'Family
_Hist_2', 'Family_Hist_3', 'Family_Hist_4', 'Family_Hist_5', 'Medical_History_1', 'Medical_History_2', 'Medical_Histo
ry_3', 'Medical_History_4', 'Medical_History_5', 'Medical_History_6', 'Medical_History_7', 'Medical_History_8', 'Medi
cal_History_9', 'Medical_History_10', 'Medical_History_11', 'Medical_History_12', 'Medical_History_13', 'Medical_Hist
ory_14', 'Medical_History_15', 'Medical_History_16', 'Medical_History_17', 'Medical_History_18', 'Medical_History_19
', 'Medical_History_20', 'Medical_History_21', 'Medical_History_22', 'Medical_History_23', 'Medical_History_24', 'Med
ical_History_25', 'Medical_History_26', 'Medical_History_27', 'Medical_History_28', 'Medical_History_29', 'Medical_Hi
story_30', 'Medical_History_31', 'Medical_History_32', 'Medical_History_33', 'Medical_History_34', 'Medical_History_3
5', 'Medical_History_36', 'Medical_History_37', 'Medical_History_38', 'Medical_History_39', 'Medical_History_40', 'Me
dical_History_41', 'Product_Info_1', 'Product_Info_3', 'Product_Info_4', 'Product_Info_5', 'Product_Info_6', 'Product
_Info_7', 'Product_Info_2_2', 'Product_Info_2_1D', 'Product_Info_2_1A', 'Product_Info_2_1E', 'Product_Info_2_1C', 'Pr
oduct_Info_2_1B']
```

Нормализация данных

Ввод [5]:
```
scaler = preprocessing.StandardScaler()
data_transformed = pd.DataFrame(scaler.fit_transform(pd.DataFrame(data,
                                                                  columns=columns)))
columns_transformed = data_transformed.columns
data_transformed["Response"] = data["Response"]
```

Оптимизация памяти

Ввод [6]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
```

```
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                            / start_mem, 1), '%)')
    return df
```

Ввод [7]:
```
data_transformed = reduce_mem_usage(data_transformed)
print (data_transformed.info())
```

```
Потребление памяти меньше на 40.49 Мб (минус 75.1 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 119 entries, 0 to Response
dtypes: float16(118), int8(1)
memory usage: 13.4 MB
None
```

Разделение данных

Преобразуем выборки в отдельные наборы данных

Ввод [8]:
```
data_train, data_test = train_test_split(data_transformed,
                                         test_size=0.2)
data_train = pd.DataFrame(data_train)
data_test = pd.DataFrame(data_test)
print (data_train.head())
```

```
              0         1         2         3         4         5         6  \
49512 -0.513672 -0.710449 -0.391602 -0.178589 -1.634766 -0.169434  0.862305
4432  -0.866211 -0.220581 -1.981445 -0.935547  0.611816 -0.169434 -1.159180
6493  -1.688477 -1.445312 -0.845703 -1.436523 -1.634766 -0.169434  0.862305
42525 -1.359375 -0.465576  0.213989 -1.457031  0.611816  5.902344  0.862305
12794  0.308838 -0.220581 -0.164429  0.578613  0.611816 -0.169434 -1.159180

              7         8         9  ...       109       110       111  \
49512 -1.013672  0.864258 -0.928711  ... -0.083679  0.441650 -0.149292
4432   1.100586 -1.156250  1.130859  ... -0.083679  0.441650 -0.149292
6493  -1.013672  0.863770 -0.928711  ... -0.083679 -2.263672 -0.149292
42525  0.043671  0.861328 -0.928711  ... -0.083679  0.441650 -0.149292
12794  1.100586 -1.156250  1.130859  ... -0.083679  0.441650 -0.149292

            112       113       114       115       116       117  Response
49512 -0.666992 -1.332031 -0.623535 -0.215942 -0.128906  7.035156         8
4432   0.733398 -1.332031  1.604492 -0.215942 -0.128906 -0.142090         8
6493   0.266846  0.750977 -0.623535 -0.215942 -0.128906 -0.142090         7
42525  0.266846  0.750977 -0.623535 -0.215942 -0.128906 -0.142090         6
12794 -0.200073  0.750977 -0.623535 -0.215942 -0.128906 -0.142090         6

[5 rows x 119 columns]
```

Перекрестная проверка случайного леса

Каждое дерево (по умолчанию, их 100) строится на своей части выборки со своим набором параметров (max_features). Решение принимается путем голосования деревьев.

Например, 10 деревьев для 1 строки (кортежа) исходных параметров дали следующие классы и их вероятности:

{1:0.5, 1:0.8, 2:0.9, 3:0.7, 5:0.5, 1:0.4, 2:0.5, 6:0.5, 3:0.4, 1:0.95}

По итогам голосования выбирается самый популярный класс, это 1 в данном случае.

Если в случайном лесу слишком много деревьев, то точность предсказания будет меньше, чем у одного, полностью обученного дерева. Число деревьев (estimators) должно соответствовать количеству классов в предсказании (class), размеру выборки (N) и числу разбиений (fold). Примерная формула: estimators = N / (20-100) / fold / class

В нашем случае, N=60000, fold=5, class=8 => estimators=15...75

Ввод [9]:
```
x = pd.DataFrame(data_train, columns=columns_transformed)
model = RandomForestClassifier(random_state=17, n_estimators=77,
                               max_depth=17, max_features=27, min_samples_leaf=30)
```

Диапазон тестирования параметров модели ограничен только вычислительной мощностью. Для проверки модели имеет смысл провести индивидуальные перекрестные проверки для каждого параметра в отдельности, затем в итоговой проверке перепроверить самые лучшие найденные параметры с отклонением +/-10%.

Ввод [11]:
```
tree_params = {
    'max_depth': range(15,17),
    'max_features': range(26,28),
    'n_estimators': range(75,77),
    'min_samples_leaf': range(19,21)
}
tree_grid = GridSearchCV(model, tree_params, cv=5, n_jobs=2,
          verbose=True, scoring=make_scorer(cohen_kappa_score))
tree_grid.fit(x, data_train["Response"])
```

Out[11]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=17,
                                              max_features=27,
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=30,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=77, n_jobs=None,
                                              oob_score=False, random_state=17,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=2,
             param_grid={'max_depth': range(15, 17),
                         'max_features': range(26, 28),
                         'min_samples_leaf': range(19, 21),
                         'n_estimators': range(75, 77)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(cohen_kappa_score), verbose=True)
```

Выведем самые оптимальные параметры и построим итоговую модель

Ввод [13]:
```
print (tree_grid.best_params_)
model = RandomForestClassifier(random_state=17,
        min_samples_leaf=tree_grid.best_params_['min_samples_leaf'],
        max_features=tree_grid.best_params_['max_features'],
        max_depth=tree_grid.best_params_['max_depth'],
        n_estimators=tree_grid.best_params_['n_estimators'])
model.fit(x, data_train["Response"])
```

```
{'max_depth': 15, 'max_features': 27, 'min_samples_leaf': 19, 'n_estimators': 75}
```

Out[13]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=15, max_features=27,
```

```
max_leaf_nodes=None, max_samples=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=19, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=75,
n_jobs=None, oob_score=False, random_state=17, verbose=0,
warm_start=False)
```

Предсказание данных и оценка модели

Ввод [14]:
```
x_test = pd.DataFrame(data_test, columns=columns_transformed)
data_test["target"] = model.predict(x_test)
```

Кластеризация дает 0.192, kNN(100) - 0.3, лог. регрессия - 0.512/0.496, SVM - 0.95, реш. дерево - 0.3

Ввод [15]:
```
print ("Случайный лес:",
       round(cohen_kappa_score(data_test["target"],
             data_test["Response"], weights="quadratic"), 3))
```

```
Случайный лес: 0.487
```

Матрица неточностей

Ввод [16]:
```
print ("Случайный лес\n",
       confusion_matrix(data_test["target"], data_test["Response"]))
```

```
Случайный лес
 [[ 115   63   11   18   12   32    8    1]
 [ 192  303   12    0   68   66    6    4]
 [  20   20   52   10    1    2    0    0]
 [  42   38   51  189    0    2    0    3]
 [  95  161   22    0  581   92   14    6]
 [ 321  272   25   33  219 1234  343  133]
 [ 148  132    2    8   56  270  614  132]
 [ 324  308    7   42  122  524  648 3648]]
```

БУСТИНГ С XGBOOST

Постановка задачи

Загрузим данные, приведем их к числовым, заполним пропуски, нормализуем данные и оптимизируем память.

Разделим выборку на обучающую/проверочную в соотношении 80/20.

Построим XGBoost модель.

Проведем предсказание и проверим качество через каппа-метрику.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix, make_scorer
from sklearn.model_selection import GridSearchCV, cross_val_score
from xgboost import XGBClassifier
from sklearn import preprocessing
```

Загрузка данных

Ввод [2]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Предобработка данных

Ввод [3]:
```
data["Product_Info_2_1"] = data["Product_Info_2"].str.slice(0, 1)
data["Product_Info_2_2"] = pd.to_numeric(data["Product_Info_2"].str.slice(1, 2))
data.drop(labels=["Product_Info_2"], axis=1, inplace=True)
for l in data["Product_Info_2_1"].unique():
    data["Product_Info_2_1" + l] = data["Product_Info_2_1"].isin([l]).astype("int8")
data.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
data.fillna(value=-1, inplace=True)
data["Response"] = data["Response"] - 1
```

Набор столбцов для расчета ¶

Ввод [4]:
```
columns_groups = ["Insurance_History", "InsuredInfo", "Medical_Keyword",
                  "Family_Hist", "Medical_History", "Product_Info"]
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
for cg in columns_groups:
    columns.extend(data.columns[data.columns.str.startswith(cg)])
print (columns)
```

```
['Wt', 'Ht', 'Ins_Age', 'BMI', 'Insurance_History_1', 'Insurance_History_2', 'Insurance_History_3', 'Insurance_History_4', 'Insurance_History_5', 'Insurance_History_7', 'Insurance_History_8', 'Insurance_History_9', 'Medical_Keyword_1', 'Medical_Keyword_2', 'Medical_Keyword_3', 'Medical_Keyword_4', 'Medical_Keyword_5', 'Medical_Keyword_6', 'Medical_Keyword_7', 'Medical_Keyword_8', 'Medical_Keyword_9', 'Medical_Keyword_10', 'Medical_Keyword_11', 'Medical_Keyword_12', 'Medical_Keyword_13', 'Medical_Keyword_14', 'Medical_Keyword_15', 'Medical_Keyword_16', 'Medical_Keyword_17', 'Medical_Keyword_18', 'Medical_Keyword_19', 'Medical_Keyword_20', 'Medical_Keyword_21', 'Medical_Keyword_22', 'Medical_Keyword_23', 'Medical_Keyword_24', 'Medical_Keyword_25', 'Medical_Keyword_26', 'Medical_Keyword_27', 'Medical_Keyword_28', 'Medical_Keyword_29', 'Medical_Keyword_30', 'Medical_Keyword_31', 'Medical_Keyword_32', 'Medical_Keyword_33', 'Medical_Keyword_34', 'Medical_Keyword_35', 'Medical_Keyword_36', 'Medical_Keyword_37', 'Medical_Keyword_38', 'Medical_Keyword_39', 'Medical_Keyword_40', 'Medical_Keyword_41', 'Medical_Keyword_42', 'Medical_Keyword_43', 'Medical_Keyword_44', 'Medical_Keyword_45', 'Medical_Keyword_46', 'Medical_Keyword_47', 'Medical_Keyword_48', 'Family_Hist_1', 'Family_Hist_2', 'Family_Hist_3', 'Family_Hist_4', 'Family_Hist_5', 'Medical_History_1', 'Medical_History_2', 'Medical_History_3', 'Medical_History_4', 'Medical_History_5', 'Medical_History_6', 'Medical_History_7', 'Medical_History_8', 'Medical_History_9', 'Medical_History_10', 'Medical_History_11', 'Medical_History_12', 'Medical_History_13', 'Medical_History_14', 'Medical_History_15', 'Medical_History_16', 'Medical_History_17', 'Medical_History_18', 'Medical_History_19', 'Medical_History_20', 'Medical_History_21', 'Medical_History_22', 'Medical_History_23', 'Medical_History_24', 'Medical_History_25', 'Medical_History_26', 'Medical_History_27', 'Medical_History_28', 'Medical_History_29', 'Medical_History_30', 'Medical_History_31', 'Medical_History_32', 'Medical_History_33', 'Medical_History_34', 'Medical_History_35', 'Medical_History_36', 'Medical_History_37', 'Medical_History_38', 'Medical_History_39', 'Medical_History_40', 'Medical_History_41', 'Product_Info_1', 'Product_Info_3', 'Product_Info_4', 'Product_Info_5', 'Product_Info_6', 'Product_Info_7', 'Product_Info_2_2', 'Product_Info_2_1D', 'Product_Info_2_1A', 'Product_Info_2_1E', 'Product_Info_2_1C', 'Product_Info_2_1B']
```

Нормализация данных

Ввод [5]:
```
scaler = preprocessing.StandardScaler()
data_transformed = pd.DataFrame(scaler.fit_transform(pd.DataFrame(data,
                                                                  columns=columns)))
columns_transformed = data_transformed.columns
data_transformed["Response"] = data["Response"]
```

Оптимизация памяти

Ввод [6]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                             / start_mem, 1), '%)')
    return df
```

Ввод [7]:
```
data_transformed = reduce_mem_usage(data_transformed)
print (data_transformed.info())
```

```
Потребление памяти меньше на 40.49 Мб (минус 75.1 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 119 entries, 0 to Response
dtypes: float16(118), int8(1)
memory usage: 13.4 MB
None
```

Разделение данных

Преобразуем выборки в отдельные наборы данных

Ввод [8]:
```
data_train, data_test = train_test_split(data_transformed,
                                         test_size=0.2)
data_train = pd.DataFrame(data_train)
data_test = pd.DataFrame(data_test)
print (data_train.head())
```

```
              0         1         2         3         4         5         6  \
36608 -0.748535 -0.710449 -0.921387 -0.500488  0.611816 -0.169434 -1.159180
25843 -1.923828 -2.423828  1.424805 -1.285156  0.611816 -0.169434  0.862305
2498  -0.395996 -0.220581  0.743652 -0.329834 -1.634766 -0.169434  0.862305
38768 -1.335938 -1.200195 -1.527344 -1.044922  0.611816 -0.169434 -1.159180
40281  1.295898  0.269287  0.970703  1.440430 -1.634766 -0.169434  0.862305
```

```
             7         8         9  ...       109       110       111  \
36608  1.100586 -1.156250  1.130859  ... -0.083679  0.441650 -0.149292
25843 -1.013672  0.874512 -0.928711  ... -0.083679  0.441650 -0.149292
2498  -1.013672  0.864746 -0.928711  ... -0.083679 -2.263672 -0.149292
38768  1.100586 -1.156250  1.130859  ... -0.083679  0.441650 -0.149292
40281 -1.013672  0.861328 -0.928711  ... -0.083679  0.441650 -0.149292

            112       113       114       115       116       117  Response
36608  0.266846  0.750977 -0.623535 -0.215942 -0.128906 -0.142090         7
25843  2.134766 -1.332031  1.604492 -0.215942 -0.128906 -0.142090         1
2498  -1.133789  0.750977 -0.623535 -0.215942 -0.128906 -0.142090         5
38768 -0.666992 -1.332031 -0.623535 -0.215942 -0.128906  7.035156         7
40281 -1.133789  0.750977 -0.623535 -0.215942 -0.128906 -0.142090         5

[5 rows x 119 columns]
```

XGBoost = градиентный бустинг деревьев решений

Строится решающее дерево, проверяется, после этого строится новое дерево, которое исправляет ошибки первого (используется последовательное применение моделей). И так - несколько итераций или до достижения нужной точности предсказания.

Основное отличие XGBoost от обычного градиентного бустинга - в использовании второй производной ошибки, из-за этого сходимость ансамбля лучше. Также XGBoost позволяет распараллелить вычисления.

Будем использовать параметры случайного леса для модели.

Ввод [9]:
```
x = pd.DataFrame(data_train, columns=columns_transformed)
model = XGBClassifier(max_depth=17, max_features=27,
                      n_estimators=76, min_samples_leaf=20)
```

Построим итоговую модель

Ввод [10]:
```
model.fit(x, data_train["Response"])
```

Out[10]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=17,
              max_features=27, min_child_weight=1, min_samples_leaf=20,
              missing=None, n_estimators=76, n_jobs=1, nthread=None,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
              subsample=1, verbosity=1)
```

Предсказание данных и оценка модели

Ввод [11]:
```
x_test = pd.DataFrame(data_test, columns=columns_transformed)
data_test["target"] = model.predict(x_test)
```

Кластеризация дает 0.192, kNN(100) - 0.3, лог. регрессия - 0.512/0.496, SVM - 0.95, реш. дерево - 0.3, случайный лес - 0.487

Ввод [12]:
```
print ("XGBoost:",
       round(cohen_kappa_score(data_test["target"],
             data_test["Response"], weights="quadratic"), 3))
```

```
XGBoost: 0.538
```

Матрица неточностей

Ввод [13]:
```
print ("XGBoost\n",
       confusion_matrix(data_test["target"], data_test["Response"]))
```

```
XGBoost
 [[ 245  176   19   12   50   93   28   18]
 [ 176  315   14    6   75   78   26   19]
 [  27   24   97   19    0    1    0    0]
 [  28   25   46  199    0    3    1    1]
 [  82  166    4    0  564   83    7    8]
 [ 240  269   12   16  204 1241  251  103]
 [ 158  129    8    7   70  327  716  200]
 [ 263  260    7   33   93  480  560 3495]]
```

Часть 6
ПРОДВИНУТЫЕ АНСАМБЛИ

LIGHTGBM

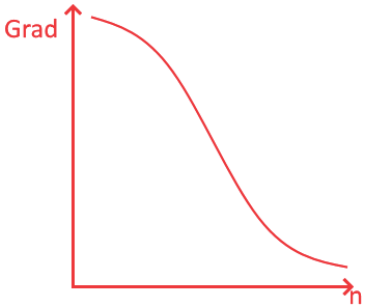
LightGBM (Light Gradient Boosting Machine) – это подход (точнее библиотека машинного обучения, предложенная компанией «Майкрософт» в области градиентного бустинга.

Он по точности сравним с Extreme Gradient Boosting, уступая стохастическому градиентному бустингу[2].

Однако, он работает существенно быстрее. «Лёгкость» отражена даже в его названии. Давайте подробнее разберём отличия:

Первый подход, который используется в LightGBM – это GOSS (от английского «упорядоченная по градиенту выборка»). Алгоритм рассмотрения элементов состоит в том, что предварительно мы отсортируем элементы по градиенту (наименее характерные и наиболее характерные по некоторому возрастанию градиента).

То есть GOSS = сортировка данных по градиенту.

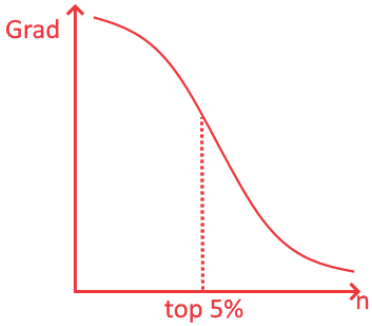


Поскольку мы хотим, с одной стороны, обучиться и охватить моделью всё возможное пространство данных, но, с другой стороны, при этом снизив её переобучение, то принято правило, по которому в первую очередь, рассматриваются те элементы, которые *наименее похожи* на всех остальных (чтобы модель их могла хорошо предсказывать). А поскольку похожие друг на друга элементы не сильно отличаются друг от друга, то модель точно обучится на них без переобучения с надлежащей точностью.

Иными словами, GOSS трактует собственный сценарий выборки: это ситуация, при которой мы выберем небольшое количество элементов не похо-

[2] В большинстве задач. Но вот на категориальных данных он может показать равную точность.

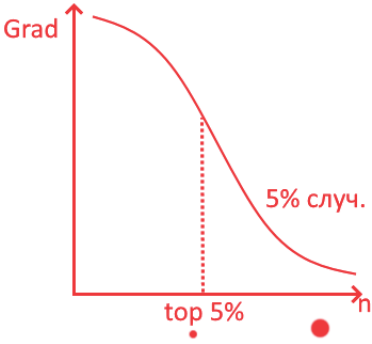
жих друг на друга (top, обычно выбирается по процентилю от выборки, например top 5 %)



Далее, выбрав 1/20 наименее характерных элементов, добираем к ней ещё сколько-то элементов (добираем до необходимого количества) случайным образом из остальных 95 %.

Например, нам необходимо 10 %, тогда мы выбираем 5 % случайным образом. Важно, чтобы эти элементы представляли выборку репрезентативно, то есть в нашем случае из 95 % выбрано ещё 5 %, тогда каждый из этих элементов представляет собой 19 других.

Когда мы, например, строим дерево решений, то разбиваем его по энтропии. В таком случае, веса выбранных элементов тоже перераспределяются:



Менее характерные элементы мы принимаем весом в 1, а более характерные (выбранные случайным образом) за 19.

За счет того, что мы взяли всего 10 % от тех элементов, разумеется, что деревья будут строиться примерно в 10 раз быстрее. Именно по этой причине LightGBM работает так быстро.

Другой важной отличительной особенностью является то, как он работает с категориями и с переменами, которые взаимно исключаются. Напом-

ним, что категории мы обычно представляем в виде единичных векторов. Взаимно исключающие параметры позволяют существенно уменьшить пространство категориальных параметров. Кроме того, что мы выбираем в 10 раз меньше элементов, дополнительно мы можем и параметров выбирать на 20, 30 или даже 50 % меньше. Этот подход делает LightGBM ещё быстрее.

И третий момент, который немного добавляет LightGBM точность, чтобы он был сравним с Extreme GBM, дерево строят не по уровням, а по параметрам. Это позволяет сохранять сбалансированность дерева, не теряя при этом в совокупности хранимой деревом информации. Из-за балансировки дерева увеличивается время для решения задачи классификации, однако классификация – это не сильная сторона LightGBM.

С учетом всех перечисленных преимуществ, LightGBM имеет смысл применять тогда, когда нам нужно, например, оперативно построить модель градиентного бустинга по большому датасету (или в принципе понять работает ли градиентный бустинг для данной задачи или нет). LightGBM даст нам после оптимизации гиперпараметров примерно максимальную точность градиентного бустинга (он может быть улучшен и ещё, но уже не так сильно).

Также LightGBM может быть использован для примерной оптимизации (или подбора) гиперпараметров, потому что дерево, которое он строит очень близко к идеальному дереву (даже для стохастического и экстремального градиентного бустинга). То есть дерево, построенное LightGBM, не сильно изменится, если мы используем полный глубокий поиск по всем параметрам, по всем измерениям, по всем энтропиям.

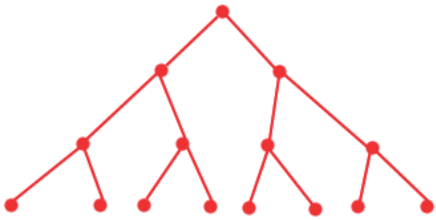
CATBOOST

Следующий продвинутой моделью градиентного бустинга является CatBoost. Однако, название несёт в себе смысл не про кошек, а про категории.

Библиотека, а точнее подход – ряд подходов, дополнительных эвристик к улучшению градиентного бустинга был предложен сотрудниками из «Яндекс». Они на основе своего долгого опыта работы с категориальными данными смогли предложить что-то, что существенно улучшило принятие решения в модели классификации на основании разнородных, каким-то непонятным образом разбросанных (плохо организованных) данных.

Какой подход они предложили и в чем его кардинальное отличие от других? Первое не основное отличие в том, что дерево CatBoost (дерево принятия решений для классификации) строится сбалансированным. Это позволяет нам максимально быстро принимать решения, то есть скорость работы и скорость обучения является фишкой. CatBoost обучается одним из самых быстрых среди всех библиотек и решения принимает тоже очень быстро (на уровне случайного леса и сверхслучайных деревьев).

Итак, дерево в CatBoost строится сбалансированным.



Как строится дерево принятия решений, которое в дальнейшем улучшается (разумеется, по методу бустинга)? Мы берем случайным образом сформированные последовательности из нашей выборки (предварительно перемешиваем их много-много раз). Потом мы берем условно 10 или 20 таких перемешанных последовательностей, берем из них несколько первых элементов (получаем некоторое подобие бэггинга), и на основании этих организованных подвыборок из нашей обучающей выборки данных, мы строим дерево, но не конечное дерево, а скажем так скелет дерева.

Далее этот скелет мы заполняем параметрами, по которым мы хотели бы разбить значения (делаем некоторую примерку – средневзвешенную прикидку по нашим выборкам из данных). На основании бэггинга мы строим скелет дерева и затем его заполняем еще одной, также рандомизированной выборкой, - тоже не полностью, но одной. То есть на основе одной выборки мы заполняем конкретные значения параметров, по которым идет разбиение. На основании многих выборок усредненного значения мы делаем скелет, по которому дальше будем принимать решения (то есть конкретные точки принятия решений заполняются на основе одной выборки, скелет делается на основании многих выборок). Это позволяет использовать некоторый «голос большинства» для того, чтобы выделить наиболее важные существенные фичи (набор параметров или параметры, которые можно использовать).

Также CatBoost отлично работает с категориальными данными. Подход примерно такой же, как и в LightGBM, то есть мы категории рассматриваем как набор каких-то значений (от 0 и до какого-то числа) и разбиваем по этому набору значений. По этому набору значений на каждом уровне дерева у нас может быть несколько дальше разбиений.

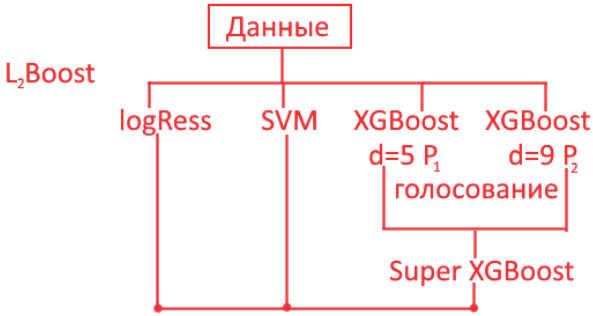
Важная отличительная особенность CatBoost от остальных в том, что к рассмотрению принимаются мономы. То есть произведения категориальных переменных друг на друга (пары значений разных категорий). Это позволяет еще улучшить работу алгоритма классификации в случае использования категориальных переменных, то есть нам не нужно в наши исходные данные добавлять все возможные произведения категорий, и мы можем положиться на CatBoost. Он непременно справится с поставленной задачей и очень быстро отработает данные.

АНСАМБЛЬ СТЕКИНГА

Ансамбль стекинга представляет собой некоторый венец, корону, лучшее, что смогло придумать человечество в борьбе за точность решения задачи машинного обучения.

В чем заключается ансамбль стекинга и как он строится? В отличие от бэггинга и бустинга, стекинг имеет принципиальное отличие в том, что строится на разнородных моделях. Предположим, что у нас есть некоторый набор данных: мы провели какую-то диагностику и на этом наборе данных использовали оптимизацию гиперпараметров, K-Fold-разбиение и так далее. В итоге получили, что у нас есть, например, XGBoost хорошо отработал, у нас хорошо отработал SVM и у нас хорошо отработала логистическая регрессия. У нас есть три алгоритма, но, поскольку XGBoost – это ансамбль, так как он оперирует деревьями как базовыми моделями; SVM – это модель; логистическая регрессия у нас тоже может быть ансамблем, если, например, на неё применили L_2Boost.

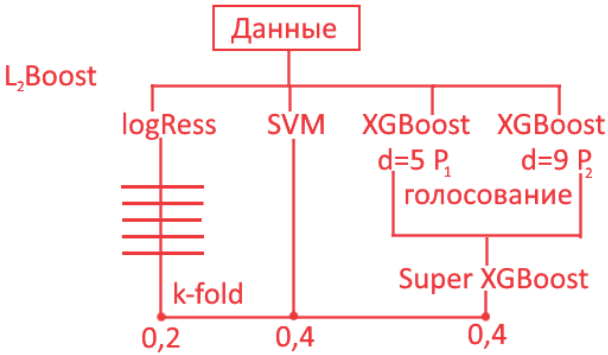
У нас уже есть некоторые, даже уже ансамбли могут быть на этом этапе, однако, подчеркнём, что они *разнородные* (они используют разный математический аппарат). Так вот, если у нас есть набор однородных моделей, мы можем их объединить опять-таки с помощью стекинга, объединяем в некоторый ансамбль дополнительно следующий. И уже его, скорее всего, мы будем использовать как вход следующего уровня принятия решения.



Если сложных моделей несколько, то применяется голосование. голосование является базовым объединяющим алгоритмом для ансамбля стекинга (мы можем просто собрать модели, прогнать их через наши проверочные данные, и это уже будет ансамбль стекинга).

Однако, сила ансамбля стекинга немножко в другом. Во-первых, мы можем обучать данные и проверять их на всем массиве данных, используя кроссвалидацию K-Fold – подхода. При ней мы берем некоторую среднюю точность наших моделей, но при этом они обучены на всем наборе наших данных. Кроме того, что они обучены на всем наборе данных, они и проверены на всем наборе наших данных независимо.

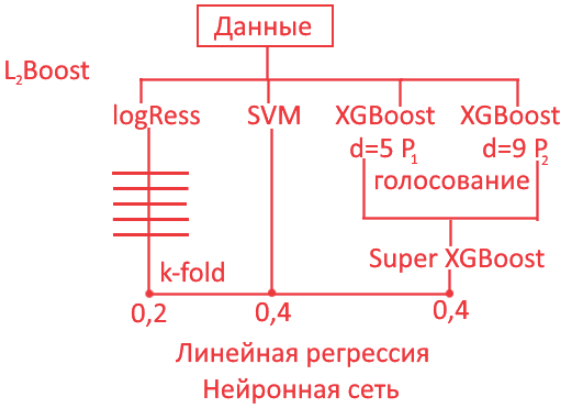
Дальше мы можем оценить точность работы каждой из моделей и сказать, что, если какая-то модель ошибается больше, то мы дадим ей меньший коэффициент. Допустим, логистическая регрессия работает хуже, поэтому ей присвоим коэффициент 0.2, а остальным по 0,4.



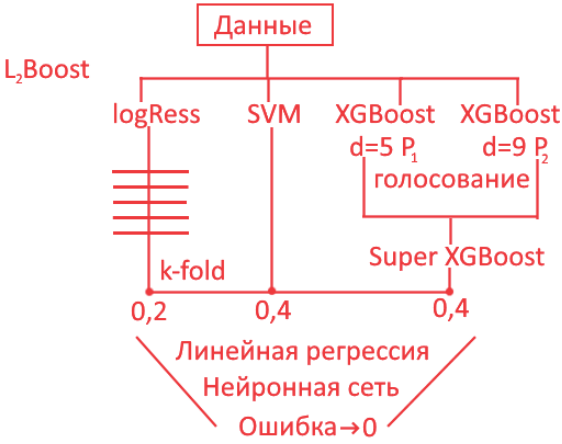
В таком случае у нас будет уже некоторое взвешенное голосование. Это следующий этап, который мы могли бы здесь придумать и который реализуется в ансамблях стекинга.

Еще один вариант состоит в том, что после того, как мы проверили независимо через K-Fold кроссвалидацию наши модели, мы можем узнать какие классы каждая из моделей предсказывает лучше, а какие хуже. И можем, например, оставить для предсказания модели только те классы, с которыми она справляется лучше остальных на независимой проверке в K-Fold кроссвалидации. Это еще один подход, так как мы выбираем уже не веса усредняем, а среднюю ошибку. Этот вариант подхода линейный (как и предыдущие).

Также на выходе мы можем поставить линейную регрессию и оптимизировать работу наших моделей на всех наших данных (кроме линейной регрессии может быть использована и нейронная сеть).



На вход нейронной сети подаем предсказания наших моделей (или их ошибки) на всех данных (эти ошибки должны быть независимыми!). Нейронная сеть может сказать каким образом нам нужно использовать комбинацию предсказаний наших всех моделей и ансамблей моделей, и даже стекинговых ансамблей моделей и как их нам нужно скомбинировать, чтобы добиться минимальной ошибки при максимальной точности нашего предсказания. Нейронная сеть сводит общую ошибку к 0.



Сведение ошибки к нулю на независимых предсказаниях данных может выполняться любой уже моделью на выходе и это тоже будет ансамблем стекинга.

Ансамбль стекинга объединяет множество *разнородных* (это ключевое отличие от других ансамблей) моделей и позволяет за счет различного комби-

нирования предсказаний этих моделей существенно повысить обобщающую силу всего ансамбля.

Несмотря на то, что концепция звучит крайне привлекательно, существуют и недостатки. Плохая новость заключается в том, что если у нас какая-то модель, например, SuperXGBoost работает существенно лучше остальных, то, когда мы все это объединим, получим точность почти этой модели – SuperXGBoost (может быть немногим лучше).

Ансамбль стекинга в случае хорошо работающих моделей, которые предсказывают последовательно (то есть просто покрывают ошибки друг друга), примерно ничего не даёт: точность очень улучшается крайне низко.

Соответственно, ансамбль стекинга хорошо работает, когда у нас модели по-разному (именно по-разному!) предсказывают данные, убирают разные параметры, для того чтобы эти данные предсказать, и как бы «с разных сторон ощупывают слона».

Аналогию можно продолжить в том понимании, что хорошо работающие алгоритмы, будучи объединенными в стекинга, просто посоревнуются между собой, кто быстрее доберется «от хобота до хвоста». Это не имеет никакого смысла, потому что лучшую модель можно выбрать сразу.

В случае плохо работающих алгоритмов, разные методы могут по-разному «обходить слона», то есть некоторые будут идти сверху вниз, а другие слева направо, имея между друг другом малейшие пересечения. В ансамбле стекинга картина сложится целиком и будет наиболее полной. Именно в использование разнородных моделей и состоит ключевое качество стекинга – понять задачу с разных сторон при использовании разных алгоритмов. В случае, когда ситуация сложилась наилучшим образом, на выходе будет самая отличная модель, позволяющая с высокой точностью обрабатывать тестовые данные. Однако, на практике такое получается не всегда.

ЧАСТЬ ПРАКТИЧЕСКИХ НАВЫКОВ К 6

LIGHTGBM

Постановка задачи

Загрузим данные, приведем их к числовым, заполним пропуски, нормализуем данные и оптимизируем память.

Разделим выборку на обучающую/проверочную в соотношении 80/20.

Построим последовательный ансамбль решающих деревьев, используя облегченный градиентный бустинг (LightGBM). Используем перекрестную проверку, чтобы найти наилучшие параметры ансамбля.

Проведем предсказание и проверим качество через каппа-метрику.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix, make_scorer
import lightgbm as lgb
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn import preprocessing
```

Загрузка данных

Ввод [2]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Предобработка данных

Дополнительно преобразуем значение класса: начнем его с нуля.

Ввод [3]:
```
data["Product_Info_2_1"] = data["Product_Info_2"].str.slice(0, 1)
data["Product_Info_2_2"] = pd.to_numeric(data["Product_Info_2"].str.slice(1, 2))
data.drop(labels=["Product_Info_2"], axis=1, inplace=True)
for l in data["Product_Info_2_1"].unique():
    data["Product_Info_2_1" + l] = data["Product_Info_2_1"].isin([l]).astype("int8")
data.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
data.fillna(value=-1, inplace=True)
data["Response"] = data["Response"] - 1
```

Набор столбцов для расчета

Ввод [4]:
```
columns_groups = ["Insurance_History", "InsuredInfo", "Medical_Keyword",
                  "Family_Hist", "Medical_History", "Product_Info"]
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
for cg in columns_groups:
    columns.extend(data.columns[data.columns.str.startswith(cg)])
print (columns)
```

```
['Wt', 'Ht', 'Ins_Age', 'BMI', 'Insurance_History_1', 'Insurance_History_2', 'Insurance_History_3', 'Insurance_History_4', 'Insurance_History_5', 'Insurance_History_7', 'Insurance_History_8', 'Insurance_History_9', 'Medical_Keyword_1', 'Medical_Keyword_2', 'Medical_Keyword_3', 'Medical_Keyword_4', 'Medical_Keyword_5', 'Medical_Keyword_6', 'Medical_Keyword_7', 'Medical_Keyword_8', 'Medical_Keyword_9', 'Medical_Keyword_10', 'Medical_Keyword_11', 'Medical_Keyword_12', 'Medical_Keyword_13', 'Medical_Keyword_14', 'Medical_Keyword_15', 'Medical_Keyword_16', 'Medical_Keyword_17', 'Medical_Keyword_18', 'Medical_Keyword_19', 'Medical_Keyword_20', 'Medical_Keyword_21', 'Medical_Keyword_22', 'Medical_Keyword_23', 'Medical_Keyword_24', 'Medical_Keyword_25', 'Medical_Keyword_26', 'Medical_Keyword_27', 'Medical_Keyword_28', 'Medical_Keyword_29', 'Medical_Keyword_30', 'Medical_Keyword_31', 'Medical_Keyword_32', 'Medical_Keyword_33', 'Medical_Keyword_34', 'Medical_Keyword_35', 'Medical_Keyword_36', 'Medical_Keyword_37', 'Medical_Keyword_38', 'Medical_Keyword_39', 'Medical_Keyword_40', 'Medical_Keyword_41', 'Medical_Keyword_42', 'Medical_Keyword_43', 'Medical_Keyword_44', 'Medical_Keyword_45', 'Medical_Keyword_46', 'Medical_Keyword_47', 'Medical_Keyword_48', 'Family_Hist_1', 'Family_Hist_2', 'Family_Hist_3', 'Family_Hist_4', 'Family_Hist_5', 'Medical_History_1', 'Medical_History_2', 'Medical_History_3', 'Medical_History_4', 'Medical_History_5', 'Medical_History_6', 'Medical_History_7', 'Medical_History_8', 'Medical_History_9', 'Medical_History_10', 'Medical_History_11', 'Medical_History_12', 'Medical_History_13', 'Medical_History_14', 'Medical_History_15', 'Medical_History_16', 'Medical_History_17', 'Medical_History_18', 'Medical_History_19', 'Medical_History_20', 'Medical_History_21', 'Medical_History_22', 'Medical_History_23', 'Medical_History_24', 'Medical_History_25', 'Medical_History_26', 'Medical_History_27', 'Medical_History_28', 'Medical_History_29', 'Medical_History_30', 'Medical_History_31', 'Medical_History_32', 'Medical_History_33', 'Medical_History_34', 'Medical_History_35', 'Medical_History_36', 'Medical_History_37', 'Medical_History_38', 'Medical_History_39', 'Medical_History_40', 'Medical_History_41', 'Product_Info_1', 'Product_Info_3', 'Product_Info_4', 'Product_Info_5', 'Product_Info_6', 'Product_Info_7', 'Product_Info_2_2', 'Product_Info_2_1D', 'Product_Info_2_1A', 'Product_Info_2_1E', 'Product_Info_2_1C', 'Product_Info_2_1B']
```

Нормализация данных

Ввод [5]:
```
scaler = preprocessing.StandardScaler()
data_transformed = pd.DataFrame(scaler.fit_transform(pd.DataFrame(data,
                                                                  columns=columns)))
columns_transformed = data_transformed.columns
data_transformed["Response"] = data["Response"]
```

Оптимизация памяти

Ввод [6]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                             / start_mem, 1), '%)')
    return df
```

Ввод [7]:
```
data_transformed = reduce_mem_usage(data_transformed)
print (data_transformed.info())
```

```
Потребление памяти меньше на 40.49 Мб (минус 75.1 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 119 entries, 0 to Response
dtypes: float16(118), int8(1)
memory usage: 13.4 MB
None
```

Разделение данных

Преобразуем выборки в отдельные наборы данных

Ввод [8]:
```
data_train, data_test = train_test_split(data_transformed,
                                         test_size=0.2)
data_train = pd.DataFrame(data_train)
data_test = pd.DataFrame(data_test)
print (data_train.head())
```

```
              0         1         2         3         4         5         6  \
36694 -0.395996 -1.200195  1.273438  0.324707 -1.634766 -0.169434  0.862305
49283 -0.043610  0.269287  1.198242 -0.187134  0.611816 -0.169434  0.862305
47506 -0.395996 -0.465576  0.667969 -0.177368 -1.634766 -0.169434  0.862305
32456  0.802246  0.024353  1.878906  1.023438 -1.634766 -0.169434  0.862305
20731 -1.100586 -0.955078 -0.315918 -0.846191  0.611816 -0.169434 -1.159180

              7         8         9  ...       109       110       111  \
36694 -1.013672  0.862305 -0.928711  ... -0.083679  0.441650 -0.149292
49283 -1.013672  0.862305 -0.928711  ... -0.083679  0.441650 -0.149292
47506 -1.013672  0.864746 -0.928711  ... -0.083679  0.441650 -0.149292
32456 -1.013672  0.862305 -0.928711  ... -0.083679  0.441650 -0.149292
20731  1.100586 -1.156250  1.130859  ... -0.083679 -2.263672 -0.149292

             112       113       114       115       116       117  Response
36694 -0.666992  0.750977 -0.623535 -0.215942 -0.128906 -0.14209         6
49283  2.134766 -1.332031  1.604492 -0.215942 -0.128906 -0.14209         6
47506 -0.200073  0.750977 -0.623535 -0.215942 -0.128906 -0.14209         6
32456 -1.133789 -1.332031 -0.623535  4.628906 -0.128906 -0.14209         1
20731 -0.200073  0.750977 -0.623535 -0.215942 -0.128906 -0.14209         7

[5 rows x 119 columns]
```

LightGBM

Основное отличие этого градиентного бустинга от предыдущих - использование сильно-разнородных (определяется разностью, гистограммой самих данных) экземпляров в выборке для формирования первоначального дерева: сначала рассматриваются все крайние, "плохие", случаи, а затем к ним "достраиваются" средние, "хорошие". Это позволяет еще быстрее минимизировать ошибку моделей.

Из дополнительных плюсов: алгоритм запускается сразу на всех ядрах процессора, это существенно ускоряет работу.

Ввод [9]:
```
x = pd.DataFrame(data_train, columns=columns_transformed)
model = lgb.LGBMRegressor(random_state=17, max_depth=18,
                          min_child_samples=19, num_leaves=34)
```

Также возможно провести классификации множества классов через LightGBM. В этом случае модель вернет вероятности принадлежности к каждому классу, возвращенные значения нужно будет дополнительно обработать через argmax, чтобы получить единственное значение класса.

Ввод [ ]:
```
'''model = lgb.LGBMRegressor(random_state=17, max_depth=17,
                          min_child_samples=18, num_leaves=34,
                          objective="multiclass", num_class=8)'''
```

Диапазон тестирования параметров модели ограничен только вычислительной мощностью. Для проверки модели имеет смысл провести индивидуальные перекрестные проверки для каждого параметра в отдельности, затем в итоговой проверке перепроверить самые лучшие найденные параметры с отклонением +/-10%.

Проверку качества по каппа метрике при оптимизации выполнить не удастся из-за нецелых значений Light GBM. Гиперпараметры оптимизации:

- max_depth - максимальная глубина деревьев,
- num_leaves - число листьев в каждом
- min_child_samples - минимальное число элементов выборке в листе

Ввод [10]:
```
lgb_params = {
    'max_depth': range(16,19),
    'num_leaves': range(34,37),
    'min_child_samples': range(17,20)
}
grid = GridSearchCV(model, lgb_params, cv=5, n_jobs=4, verbose=True)
grid.fit(x, data_train["Response"])
```

Out[10]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=LGBMRegressor(boosting_type='gbdt', class_weight=None,
                                     colsample_bytree=1.0,
                                     importance_type='split', learning_rate=0.1,
                                     max_depth=18, min_child_samples=19,
                                     min_child_weight=0.001, min_split_gain=0.0,
                                     n_estimators=100, n_jobs=-1, num_leaves=34,
                                     objective=None, random_state=17,
                                     reg_alpha=0.0, reg_lambda=0.0, silent=True,
                                     subsample=1.0, subsample_for_bin=200000,
                                     subsample_freq=0),
             iid='deprecated', n_jobs=4,
             param_grid={'max_depth': range(16, 19),
                         'min_child_samples': range(17, 20),
                         'num_leaves': range(34, 37)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)
```

Выведем самые оптимальные параметры и построим итоговую модель, используя 1000 последовательных деревьев.

Ввод [11]:
```
print (grid.best_params_)
model = lgb.LGBMRegressor(random_state=17,
    max_depth=grid.best_params_['max_depth'],
    min_child_samples=grid.best_params_['min_child_samples'],
    num_leaves=grid.best_params_['num_leaves'],
    n_estimators=1000,
    objective="multiclass", num_class=8)
```

```
{'max_depth': 18, 'min_child_samples': 17, 'num_leaves': 35}
```

Ввод [12]:
```
model.fit(x, data_train["Response"])
```

Out[12]:
```
LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
              importance_type='split', learning_rate=0.1, max_depth=18,
              min_child_samples=17, min_child_weight=0.001, min_split_gain=0.0,
              n_estimators=1000, n_jobs=-1, num_class=8, num_leaves=35,
              objective='multiclass', random_state=17, reg_alpha=0.0,
              reg_lambda=0.0, silent=True, subsample=1.0,
              subsample_for_bin=200000, subsample_freq=0)
```

Предсказание данных и оценка модели

LightGBM возвращает дробное значение класса, его нужно округлить.

Для multiclass используем argmax

Ввод [13]:
```
def calculate_model (x):
    return np.argmax(model.predict([x]))
```

Ввод [14]:
```
x_test = pd.DataFrame(data_test, columns=columns_transformed)
data_test["target"] = x_test.apply(calculate_model, axis=1,
                                   result_type="expand")
```

Кластеризация дает 0.192, kNN(100) - 0.3, лог. регрессия - 0.512/0.496, SVM - 0.95, реш. дерево - 0.3, случайный лес - 0.487, XGBoost - 0.536, градиентный бустинг - 0.56

Ввод [15]:
```
print ("LightGBM:",
       round(cohen_kappa_score(data_test["target"],
                               data_test["Response"], weights="quadratic"), 3))
```

```
LightGBM: 0.569
```

Матрица неточностей

Ввод [16]:
```
print ("LightGBM\n",
       confusion_matrix(data_test["target"], data_test["Response"]))
```

```
LightGBM
 [[ 326  158   25   14   41  107   49   34]
 [ 184  363   22    8  110  107   35   20]
 [  23   20   76   12    0    0    0    0]
 [  36   36   71  162    0    4    1    0]
 [  96  122    7    0  541   75    8    5]
 [ 231  249   14   23  222 1193  255  123]
 [ 103  141    3    6   70  314  671  176]
 [ 222  227    7   38   93  472  555 3571]]
```

CATBOOST

Постановка задачи

Загрузим данные, приведем их к числовым, заполним пропуски, нормализуем данные и оптимизируем память.

Разделим выборку на обучающую/проверочную в соотношении 80/20.

Построим ансамбль решающих деревьев, используя патентованный градиентный бустинг Яндекса (CatBoost). Используем перекрестную проверку, чтобы найти наилучшие параметры ансамбля.

Проведем предсказание и проверим качество через каппа-метрику.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix
from catboost import Pool, CatBoostClassifier
from sklearn import preprocessing
```

Загрузка данных

Ввод [2]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Предобработка данных

Ввод [3]:
```
data["Product_Info_2_1"] = data["Product_Info_2"].str.slice(0, 1)
data["Product_Info_2_2"] = pd.to_numeric(data["Product_Info_2"].str.slice(1, 2))
data.drop(labels=["Product_Info_2"], axis=1, inplace=True)
for l in data["Product_Info_2_1"].unique():
    data["Product_Info_2_1" + l] = data["Product_Info_2_1"].isin([l]).astype("int8")
data.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
data.fillna(value=-1, inplace=True)
data["Response"] = data["Response"] - 1
```

Набор столбцов для расчета

Ввод [4]:
```
columns_groups = ["Insurance_History", "InsuredInfo", "Medical_Keyword",
                  "Family_Hist", "Medical_History", "Product_Info"]
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
for cg in columns_groups:
    columns.extend(data.columns[data.columns.str.startswith(cg)])
print (columns)
```

```
['Wt', 'Ht', 'Ins_Age', 'BMI', 'Insurance_History_1', 'Insurance_History_2', 'Insurance_History_3', 'Insurance_History_4', 'Insurance_History_5', 'Insurance_History_7', 'Insurance_History_8', 'Insurance_History_9', 'Medical_Keyword_1', 'Medical_Keyword_2', 'Medical_Keyword_3', 'Medical_Keyword_4', 'Medical_Keyword_5', 'Medical_Keyword_6', 'Medical_Keyword_7', 'Medical_Keyword_8', 'Medical_Keyword_9', 'Medical_Keyword_10', 'Medical_Keyword_11', 'Medical_Keyword_12', 'Medical_Keyword_13', 'Medical_Keyword_14', 'Medical_Keyword_15', 'Medical_Keyword_16', 'Medical_Keyword_17', 'Medical_Keyword_18', 'Medical_Keyword_19', 'Medical_Keyword_20', 'Medical_Keyword_21', 'Medical_Keyword_22', 'Medical_Keyword_23', 'Medical_Keyword_24', 'Medical_Keyword_25', 'Medical_Keyword_26', 'Medical_Keyword_27', 'Medical_Keyword_28', 'Medical_Keyword_29', 'Medical_Keyword_30', 'Medical_Keyword_31', 'Medical_Keyword_32', 'Medical_Keyword_33', 'Medical_Keyword_34', 'Medical_Keyword_35', 'Medical_Keyword_36', 'Medical_Keyword_37', 'Medical_Keyword_38', 'Medical_Keyword_39', 'Medical_Keyword_40', 'Medical_Keyword_41', 'Medical_Keyword_42', 'Medical_Keyword_43', 'Medical_Keyword_44', 'Medical_Keyword_45', 'Medical_Keyword_46', 'Medical_Keyword_47', 'Medical_Keyword_48', 'Family_Hist_1', 'Family_Hist_2', 'Family_Hist_3', 'Family_Hist_4', 'Family_Hist_5', 'Medical_History_1', 'Medical_History_2', 'Medical_History_3', 'Medical_History_4', 'Medical_History_5', 'Medical_History_6', 'Medical_History_7', 'Medical_History_8', 'Medical_History_9', 'Medical_History_10', 'Medical_History_11', 'Medical_History_12', 'Medical_History_13', 'Medical_History_14', 'Medical_History_15', 'Medical_History_16', 'Medical_History_17', 'Medical_History_18', 'Medical_History_19', 'Medical_History_20', 'Medical_History_21', 'Medical_History_22', 'Medical_History_23', 'Medical_History_24', 'Medical_History_25', 'Medical_History_26', 'Medical_History_27', 'Medical_History_28', 'Medical_History_29', 'Medical_History_30', 'Medical_History_31', 'Medical_History_32', 'Medical_History_33', 'Medical_History_34', 'Medical_History_35', 'Medical_History_36', 'Medical_History_37', 'Medical_History_38', 'Medical_History_39', 'Medical_History_40', 'Medical_History_41', 'Product_Info_1', 'Product_Info_3', 'Product_Info_4', 'Product_Info_5', 'Product_Info_6', 'Product_Info_7', 'Product_Info_2_2', 'Product_Info_2_1D', 'Product_Info_2_1A', 'Product_Info_2_1E', 'Product_Info_2_1C', 'Product_Info_2_1B']
```

Нормализация данных

Ввод [5]:
```
scaler = preprocessing.StandardScaler()
data_transformed = pd.DataFrame(scaler.fit_transform(pd.DataFrame(data,
                                                                  columns=columns)))
columns_transformed = data_transformed.columns
data_transformed["Response"] = data["Response"]
```

Оптимизация памяти

Ввод [6]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
```

```
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                            / start_mem, 1), '%)')
    return df
```

Ввод [7]:
```
data_transformed = reduce_mem_usage(data_transformed)
print (data_transformed.info())
```

```
Потребление памяти меньше на 40.49 Мб (минус 75.1 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 119 entries, 0 to Response
dtypes: float16(118), int8(1)
memory usage: 13.4 MB
None
```

Разделение данных

Преобразуем выборки в отдельные наборы данных

Ввод [8]:
```
data_train, data_test = train_test_split(data_transformed,
                                         test_size=0.2)
data_train = pd.DataFrame(data_train)
data_test = pd.DataFrame(data_test)
print (data_train.head())
```

```
              0         1         2         3         4         5         6  \
29740  0.426270  1.003906  1.424805 -0.081970  0.611816 -0.169434  0.862305
53953  0.543945  1.003906  0.667969  0.049133  0.611816 -0.169434  0.862305
41278 -1.618164 -1.200195 -0.694336 -1.456055 -1.634766 -0.169434  0.862305
26944  0.097351 -0.710449 -0.618652  0.658203  0.611816 -0.169434 -1.159180
35805 -0.866211 -0.220581  0.441162 -0.935547  0.611816  5.902344 -1.159180

              7         8         9  ...       109       110       111  \
29740 -1.013672  0.863281 -0.928711  ... -0.083679  0.441650 -0.149292
53953 -1.013672  0.863770 -0.928711  ... -0.083679  0.441650 -0.149292
41278 -1.013672  0.867676 -0.928711  ... -0.083679 -2.263672 -0.149292
26944  1.100586 -1.156250  1.130859  ... -0.083679  0.441650 -0.149292
35805  1.100586 -1.156250  1.130859  ... -0.083679  0.441650 -0.149292

            112       113       114       115       116       117  Response
29740  2.134766 -1.332031  1.604492 -0.215942 -0.128906 -0.14209         6
53953 -0.666992 -1.332031  1.604492 -0.215942 -0.128906 -0.14209         5
41278 -0.200073  0.750977 -0.623535 -0.215942 -0.128906 -0.14209         7
26944  1.200195 -1.332031  1.604492 -0.215942 -0.128906 -0.14209         5
35805 -1.133789 -1.332031 -0.623535 -0.215942  7.761719 -0.14209         1

[5 rows x 119 columns]
```

CatBoost

Основные преимущества: умение работать с категориальными (номинативными) признаками и бОльшая точность, чем LighGBM

Алгоритм запускается сразу на всех ядрах процессора, это существенно ускоряет работу.

В качестве ансамблирования выберем метод опорных векторов (MVS), он ранее показал хорошую точность (и для CatBoost он тоже повышает точность на рассматриваемых данных).

Ввод [9]:
```
x = pd.DataFrame(data_train, columns=columns_transformed)
train_dataset = Pool(data=x, label=data_train["Response"])
model = CatBoostClassifier(iterations=10, learning_rate=0.57,
                           random_seed=17, depth=6, loss_function="MultiClass",
                           bootstrap_type="MVS", custom_metric="WKappa")
```

Диапазон тестирования параметров модели ограничен только вычислительной мощностью. Для проверки модели имеет смысл провести индивидуальные перекрестные проверки для каждого параметра в отдельности, затем в итоговой проверке перепроверить самые лучшие найденные параметры с отклонением +/-10%.

Гиперпараметры оптимизации:

- depth - максимальная глубина деревьев,
- learning_rate - скорость обучения
- l2_leaf_reg - L2 параметр регуляризации для функции стоимости

Ввод [10]:
```
cb_params = {
    "depth": range(5,8),
    'learning_rate': np.arange(0.56,0.59,0.01),
    'l2_leaf_reg': range(1,5),
}
cb_grid = model.grid_search(cb_params, cv=5, X=x,
                y=data_train["Response"], verbose=True)
print (cb_grid["params"])
```

```
0:      loss: 1.2816254 best: 1.2816254 (0)     total: 4.13s    remaining: 2m 24s
1:      loss: 1.2778343 best: 1.2778343 (1)     total: 7.97s    remaining: 2m 15s
2:      loss: 1.2762517 best: 1.2762517 (2)     total: 11.7s    remaining: 2m 9s
3:      loss: 1.2771619 best: 1.2762517 (2)     total: 15.4s    remaining: 2m 3s
4:      loss: 1.2856658 best: 1.2762517 (2)     total: 19.1s    remaining: 1m 58s
5:      loss: 1.2788533 best: 1.2762517 (2)     total: 22.8s    remaining: 1m 53s
6:      loss: 1.2788416 best: 1.2762517 (2)     total: 26.6s    remaining: 1m 50s
7:      loss: 1.2813785 best: 1.2762517 (2)     total: 30.5s    remaining: 1m 46s
8:      loss: 1.2885144 best: 1.2762517 (2)     total: 34.9s    remaining: 1m 44s
9:      loss: 1.2803815 best: 1.2762517 (2)     total: 38.6s    remaining: 1m 40s
10:     loss: 1.2824530 best: 1.2762517 (2)     total: 42.2s    remaining: 1m 35s
11:     loss: 1.2817148 best: 1.2762517 (2)     total: 46.1s    remaining: 1m 32s
12:     loss: 1.2697193 best: 1.2697193 (12)    total: 51.1s    remaining: 1m 30s
13:     loss: 1.2715684 best: 1.2697193 (12)    total: 56s      remaining: 1m 28s
14:     loss: 1.2709726 best: 1.2697193 (12)    total: 1m 1s    remaining: 1m 25s
15:     loss: 1.2723835 best: 1.2697193 (12)    total: 1m 5s    remaining: 1m 22s
16:     loss: 1.2690889 best: 1.2690889 (16)    total: 1m 10s   remaining: 1m 18s
17:     loss: 1.2677510 best: 1.2677510 (17)    total: 1m 15s   remaining: 1m 15s
18:     loss: 1.2716304 best: 1.2677510 (17)    total: 1m 19s   remaining: 1m 11s
19:     loss: 1.2705394 best: 1.2677510 (17)    total: 1m 24s   remaining: 1m 7s
20:     loss: 1.2726550 best: 1.2677510 (17)    total: 1m 29s   remaining: 1m 3s
21:     loss: 1.2720823 best: 1.2677510 (17)    total: 1m 34s   remaining: 60s
22:     loss: 1.2727118 best: 1.2677510 (17)    total: 1m 39s   remaining: 56.1s
23:     loss: 1.2735933 best: 1.2677510 (17)    total: 1m 44s   remaining: 52s
24:     loss: 1.2555733 best: 1.2555733 (24)    total: 1m 54s   remaining: 50.3s
25:     loss: 1.2645743 best: 1.2555733 (24)    total: 2m 4s    remaining: 47.9s
26:     loss: 1.2522148 best: 1.2522148 (26)    total: 2m 14s   remaining: 44.8s
27:     loss: 1.2617746 best: 1.2522148 (26)    total: 2m 24s   remaining: 41.3s
28:     loss: 1.2660059 best: 1.2522148 (26)    total: 2m 34s   remaining: 37.3s
29:     loss: 1.2553680 best: 1.2522148 (26)    total: 2m 46s   remaining: 33.3s
30:     loss: 1.2579626 best: 1.2522148 (26)    total: 2m 56s   remaining: 28.5s
31:     loss: 1.2669948 best: 1.2522148 (26)    total: 3m 6s    remaining: 23.4s
32:     loss: 1.2650369 best: 1.2522148 (26)    total: 3m 17s   remaining: 17.9s
33:     loss: 1.2642775 best: 1.2522148 (26)    total: 3m 27s   remaining: 12.2s
34:     loss: 1.2695645 best: 1.2522148 (26)    total: 3m 37s   remaining: 6.2s
35:     loss: 1.2579431 best: 1.2522148 (26)    total: 3m 47s   remaining: 0us
Estimating final quality...
{'depth': 7, 'l2_leaf_reg': 1, 'learning_rate': 0.5800000000000001}
```

Выведем самые оптимальные параметры и построим итоговую модель

Ввод [12]:
```
print (cb_grid["params"])
model = CatBoostClassifier(iterations=100,
        learning_rate=cb_grid["params"]["learning_rate"],
        depth=cb_grid["params"]["depth"],
        l2_leaf_reg=cb_grid["params"]["l2_leaf_reg"],
        random_seed=17, loss_function="MultiClass",
        bootstrap_type="MVS", custom_metric="WKappa")
```

```
{'depth': 7, 'l2_leaf_reg': 1, 'learning_rate': 0.5800000000000001}
```

Ввод [13]:
```
model.fit(train_dataset)
```

```
0:      learn: 1.5260121        total: 1.64s    remaining: 2m 42s
1:      learn: 1.4298948        total: 2.86s    remaining: 2m 19s
2:      learn: 1.3249085        total: 4.11s    remaining: 2m 12s
3:      learn: 1.2901868        total: 5.3s     remaining: 2m 7s
4:      learn: 1.2671400        total: 6.6s     remaining: 2m 5s
5:      learn: 1.2555368        total: 7.69s    remaining: 2m
6:      learn: 1.2450711        total: 8.91s    remaining: 1m 58s
7:      learn: 1.2376520        total: 10s      remaining: 1m 55s
8:      learn: 1.2261509        total: 11.3s    remaining: 1m 53s
9:      learn: 1.2191708        total: 12.5s    remaining: 1m 52s
10:     learn: 1.2109959        total: 13.6s    remaining: 1m 50s
11:     learn: 1.2013338        total: 14.9s    remaining: 1m 49s
12:     learn: 1.1945134        total: 16.2s    remaining: 1m 48s
13:     learn: 1.1889184        total: 17.3s    remaining: 1m 46s
14:     learn: 1.1796621        total: 18.5s    remaining: 1m 44s
15:     learn: 1.1730491        total: 19.7s    remaining: 1m 43s
16:     learn: 1.1690535        total: 20.9s    remaining: 1m 41s
17:     learn: 1.1637374        total: 22s      remaining: 1m 40s
18:     learn: 1.1583403        total: 23.2s    remaining: 1m 39s
19:     learn: 1.1521315        total: 24.5s    remaining: 1m 38s
20:     learn: 1.1469662        total: 25.7s    remaining: 1m 36s
21:     learn: 1.1425185        total: 27s      remaining: 1m 35s
22:     learn: 1.1389436        total: 28.1s    remaining: 1m 34s
23:     learn: 1.1293394        total: 29.3s    remaining: 1m 32s
24:     learn: 1.1259789        total: 30.6s    remaining: 1m 31s
25:     learn: 1.1221434        total: 31.8s    remaining: 1m 30s
26:     learn: 1.1179584        total: 33.1s    remaining: 1m 29s
27:     learn: 1.1133800        total: 34.4s    remaining: 1m 28s
28:     learn: 1.1104712        total: 35.6s    remaining: 1m 27s
29:     learn: 1.1075689        total: 36.9s    remaining: 1m 26s
30:     learn: 1.1044159        total: 38.2s    remaining: 1m 24s
31:     learn: 1.0989145        total: 39.4s    remaining: 1m 23s
32:     learn: 1.0949965        total: 40.6s    remaining: 1m 22s
33:     learn: 1.0910792        total: 41.8s    remaining: 1m 21s
34:     learn: 1.0884420        total: 43s      remaining: 1m 19s
35:     learn: 1.0850179        total: 44.3s    remaining: 1m 18s
36:     learn: 1.0802929        total: 45.5s    remaining: 1m 17s
37:     learn: 1.0766988        total: 46.7s    remaining: 1m 16s
38:     learn: 1.0726932        total: 47.9s    remaining: 1m 14s
39:     learn: 1.0643382        total: 49.2s    remaining: 1m 13s
```

```
40:     learn: 1.0609973        total: 50.4s    remaining: 1m 12s
41:     learn: 1.0576750        total: 51.6s    remaining: 1m 11s
42:     learn: 1.0543346        total: 52.7s    remaining: 1m 9s
43:     learn: 1.0525856        total: 53.9s    remaining: 1m 8s
44:     learn: 1.0489259        total: 55.1s    remaining: 1m 7s
45:     learn: 1.0455482        total: 56.3s    remaining: 1m 6s
46:     learn: 1.0431159        total: 57.9s    remaining: 1m 5s
47:     learn: 1.0406538        total: 59.2s    remaining: 1m 4s
48:     learn: 1.0379089        total: 1m       remaining: 1m 3s
49:     learn: 1.0356668        total: 1m 1s    remaining: 1m 1s
50:     learn: 1.0319510        total: 1m 3s    remaining: 1m
51:     learn: 1.0287363        total: 1m 4s    remaining: 59.6s
52:     learn: 1.0252098        total: 1m 5s    remaining: 58.3s
53:     learn: 1.0212433        total: 1m 7s    remaining: 57.1s
54:     learn: 1.0181111        total: 1m 8s    remaining: 55.9s
55:     learn: 1.0160674        total: 1m 9s    remaining: 54.6s
56:     learn: 1.0138920        total: 1m 10s   remaining: 53.4s
57:     learn: 1.0100578        total: 1m 12s   remaining: 52.2s
58:     learn: 1.0071105        total: 1m 13s   remaining: 50.9s
59:     learn: 1.0037556        total: 1m 14s   remaining: 49.6s
60:     learn: 1.0012180        total: 1m 15s   remaining: 48.3s
61:     learn: 0.9963677        total: 1m 16s   remaining: 47s
62:     learn: 0.9953593        total: 1m 17s   remaining: 45.8s
63:     learn: 0.9951924        total: 1m 19s   remaining: 44.5s
64:     learn: 0.9938143        total: 1m 20s   remaining: 43.2s
65:     learn: 0.9916066        total: 1m 21s   remaining: 41.9s
66:     learn: 0.9892000        total: 1m 22s   remaining: 40.7s
67:     learn: 0.9863433        total: 1m 23s   remaining: 39.5s
68:     learn: 0.9831088        total: 1m 25s   remaining: 38.2s
69:     learn: 0.9810235        total: 1m 26s   remaining: 37s
70:     learn: 0.9789312        total: 1m 27s   remaining: 35.7s
71:     learn: 0.9761802        total: 1m 28s   remaining: 34.4s
72:     learn: 0.9725489        total: 1m 29s   remaining: 33.1s
73:     learn: 0.9709504        total: 1m 30s   remaining: 31.9s
74:     learn: 0.9680710        total: 1m 31s   remaining: 30.6s
75:     learn: 0.9643559        total: 1m 32s   remaining: 29.3s
76:     learn: 0.9628542        total: 1m 34s   remaining: 28.1s
77:     learn: 0.9605435        total: 1m 35s   remaining: 26.8s
78:     learn: 0.9588157        total: 1m 36s   remaining: 25.6s
79:     learn: 0.9553125        total: 1m 37s   remaining: 24.3s
80:     learn: 0.9535540        total: 1m 38s   remaining: 23.1s
81:     learn: 0.9498601        total: 1m 39s   remaining: 21.9s
82:     learn: 0.9463917        total: 1m 40s   remaining: 20.7s
83:     learn: 0.9446690        total: 1m 42s   remaining: 19.4s
84:     learn: 0.9434211        total: 1m 43s   remaining: 18.2s
85:     learn: 0.9424063        total: 1m 44s   remaining: 17s
86:     learn: 0.9412346        total: 1m 45s   remaining: 15.8s
87:     learn: 0.9379394        total: 1m 46s   remaining: 14.5s
88:     learn: 0.9361016        total: 1m 47s   remaining: 13.3s
89:     learn: 0.9340184        total: 1m 48s   remaining: 12.1s
90:     learn: 0.9332158        total: 1m 50s   remaining: 10.9s
91:     learn: 0.9314344        total: 1m 51s   remaining: 9.66s
92:     learn: 0.9293507        total: 1m 52s   remaining: 8.45s
93:     learn: 0.9269872        total: 1m 53s   remaining: 7.24s
94:     learn: 0.9246659        total: 1m 54s   remaining: 6.03s
95:     learn: 0.9228249        total: 1m 55s   remaining: 4.81s
96:     learn: 0.9223726        total: 1m 56s   remaining: 3.61s
```

```
97:     learn: 0.9210276        total: 1m 57s   remaining: 2.4s
98:     learn: 0.9202147        total: 1m 58s   remaining: 1.2s
99:     learn: 0.9187761        total: 1m 59s   remaining: 0us
```

Out[13]: <catboost.core.CatBoostClassifier at 0x58f24c8>

Предсказание данных и оценка модели

Ввод [14]:
```
x_test = pd.DataFrame(data_test, columns=columns_transformed)
data_test["target"] = model.predict(Pool(data=x_test))
```

Кластеризация дает 0.192, kNN(100) - 0.3, лог. регрессия - 0.512/0.496, SVM - 0.95, реш. дерево - 0.3, случайный лес - 0.487, XGBoost - 0.536, градиентный бустинг - 0.56, LightGBM - 0.569

Ввод [15]:
```
print ("CatBoost:",
       round(cohen_kappa_score(data_test["target"],
             data_test["Response"], weights="quadratic"), 3))
```

```
CatBoost: 0.538
```

Матрица неточностей

Ввод [16]:
```
print ("CatBoost\n",
       confusion_matrix(data_test["target"], data_test["Response"]))
```

```
CatBoost
 [[ 299  188   19   13   69  120   41   31]
 [ 178  332    7    4  118  110   29   28]
 [  29   24   82   22    2    4    0    0]
 [  38   31   53  198    0    4    0    2]
 [  86  128   11    0  515  111    7    4]
 [ 240  261    8   17  203 1138  288  147]
 [ 142  108    4   10   52  272  702  177]
 [ 268  268    7   33   84  438  576 3497]]
```

АНСАМБЛЬ КЛАССИФИКАЦИИ

Постановка задачи

Загрузим данные, приведем их к числовым, заполним пропуски, нормализуем данные и оптимизируем память.

Разделим выборку на обучающую/проверочную в соотношении 80/20.

Сформируем параллельный ансамбль из логистической регрессии, SVM, случайного леса и LightGBM. Используем лучшие гиперпараметры, подобранные ранее. Итоговое решение рассчитаем на основании весов (вероятностей).

Проведем предсказание и проверим качество через каппа-метрику.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix, make_scorer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb
from sklearn import preprocessing
```

Загрузка данных

Ввод [2]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Предобработка данных

Ввод [3]:
```
data["Product_Info_2_1"] = data["Product_Info_2"].str.slice(0, 1)
data["Product_Info_2_2"] = pd.to_numeric(data["Product_Info_2"].str.slice(1, 2))
data.drop(labels=["Product_Info_2"], axis=1, inplace=True)
for l in data["Product_Info_2_1"].unique():
    data["Product_Info_2_1" + l] = data["Product_Info_2_1"].isin([l]).astype("int8")
data.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
data.fillna(value=-1, inplace=True)
data["Response"] = data["Response"] - 1
```

Набор столбцов для расчета

Ввод [4]:
```
columns_groups = ["Insurance_History", "InsuredInfo", "Medical_Keyword",
                  "Family_Hist", "Medical_History", "Product_Info"]
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
for cg in columns_groups:
    columns.extend(data.columns[data.columns.str.startswith(cg)])
print (columns)
```

```
['Wt', 'Ht', 'Ins_Age', 'BMI', 'Insurance_History_1', 'Insurance_History_2', 'Insurance_History_3', 'Insurance_History_4', 'Insurance_History_5', 'Insurance_History_7', 'Insurance_History_8', 'Insurance_History_9', 'Medical_Keyword_1', 'Medical_Keyword_2', 'Medical_Keyword_3', 'Medical_Keyword_4', 'Medical_Keyword_5', 'Medical_Keyword_6', 'Medical_Keyword_7', 'Medical_Keyword_8', 'Medical_Keyword_9', 'Medical_Keyword_10', 'Medical_Keyword_11', 'Medical_Keyword_12', 'Medical_Keyword_13', 'Medical_Keyword_14', 'Medical_Keyword_15', 'Medical_Keyword_16', 'Medical_Keyword_17', 'Medical_Keyword_18', 'Medical_Keyword_19', 'Medical_Keyword_20', 'Medical_Keyword_21', 'Medical_Keyword_22', 'Medical_Keyword_23', 'Medical_Keyword_24', 'Medical_Keyword_25', 'Medical_Keyword_26', 'Medical_Keyword_27', 'Medical_Keyword_28', 'Medical_Keyword_29', 'Medical_Keyword_30', 'Medical_Keyword_31', 'Medical_Keyword_32', 'Medical_Keyword_33', 'Medical_Keyword_34', 'Medical_Keyword_35', 'Medical_Keyword_36', 'Medical_Keyword_37', 'Medical_Keyword_38', 'Medical_Keyword_39', 'Medical_Keyword_40', 'Medical_Keyword_41', 'Medical_Keyword_42', 'Medical_Keyword_43', 'Medical_Keyword_44', 'Medical_Keyword_45', 'Medical_Keyword_46', 'Medical_Keyword_47', 'Medical_Keyword_48', 'Family_Hist_1', 'Family_Hist_2', 'Family_Hist_3', 'Family_Hist_4', 'Family_Hist_5', 'Medical_History_1', 'Medical_History_2', 'Medical_History_3', 'Medical_History_4', 'Medical_History_5', 'Medical_History_6', 'Medical_History_7', 'Medical_History_8', 'Medical_History_9', 'Medical_History_10', 'Medical_History_11', 'Medical_History_12', 'Medical_History_13', 'Medical_History_14', 'Medical_History_15', 'Medical_History_16', 'Medical_History_17', 'Medical_History_18', 'Medical_History_19', 'Medical_History_20', 'Medical_History_21', 'Medical_History_22', 'Medical_History_23', 'Medical_History_24', 'Medical_History_25', 'Medical_History_26', 'Medical_History_27', 'Medical_History_28', 'Medical_History_29', 'Medical_History_30', 'Medical_History_31', 'Medical_History_32', 'Medical_History_33', 'Medical_History_34', 'Medical_History_35', 'Medical_History_36', 'Medical_History_37', 'Medical_History_38', 'Medical_History_39', 'Medical_History_40', 'Medical_History_41', 'Product_Info_1', 'Product_Info_3', 'Product_Info_4', 'Product_Info_5', 'Product_Info_6', 'Product_Info_7', 'Product_Info_2_2', 'Product_Info_2_1D', 'Product_Info_2_1A', 'Product_Info_2_1E', 'Product_Info_2_1C', 'Product_Info_2_1B']
```

Нормализация данных

Ввод [5]:
```
scaler = preprocessing.StandardScaler()
data_transformed = pd.DataFrame(scaler.fit_transform(pd.DataFrame(data,
                                                                  columns=columns)))
columns_transformed = data_transformed.columns
data_transformed["Response"] = data["Response"]
```

Оптимизация памяти

Ввод [6]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                             / start_mem, 1), '%)')
    return df
```

Ввод [7]:
```
data_transformed = reduce_mem_usage(data_transformed)
print (data_transformed.info())
```

```
Потребление памяти меньше на 40.49 Мб (минус 75.1 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 119 entries, 0 to Response
dtypes: float16(118), int8(1)
memory usage: 13.4 MB
None
```

Разделение данных

Преобразуем выборки в отдельные наборы данных

Ввод [8]:
```
data_train, data_test = train_test_split(data_transformed,
                                         test_size=0.2)
data_train = pd.DataFrame(data_train)
data_test = pd.DataFrame(data_test)
print (data_train.head())
```

```
             0         1         2         3         4         5         6  \
16581 -0.748535 -0.710449  0.365479 -0.500488 -1.634766 -0.169434  0.862305
32658 -0.630859 -1.200195 -0.997070 -0.017761  0.611816 -0.169434  0.862305
13818 -0.748535 -0.710449  0.895020 -0.500488  0.611816 -0.169434 -1.159180
5607  -0.513672 -0.955078  1.727539 -0.016434  0.611816 -0.169434 -1.159180
54265  0.308838  1.249023  1.424805 -0.352295 -1.634766 -0.169434  0.862305
```

Построение базовых моделей

Ввод [9]:
```
x = pd.DataFrame(data_train, columns=columns_transformed)
```

SVM

Ввод [10]:
```
model_svm = SVC(kernel='linear', probability=True, max_iter=10000)
model_svm.fit(x, data_train["Response"])
```

```
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\sklearn\svm\_base.py:231: ConvergenceWarnin
g: Solver terminated early (max_iter=10000).  Consider pre-processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

Out[10]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=10000, probability=True, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Логистическая регрессия

Ввод [11]:
```
model_logr = LogisticRegression(max_iter=1000)
model_logr.fit(x, data_train["Response"])
```

Out[11]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Случайный лес

Ввод [16]:
```
model_rf = RandomForestClassifier(random_state=17, n_estimators=76,
            max_depth=17, max_features=27, min_samples_leaf=20)
model_rf.fit(x, data_train["Response"])
```

Out[16]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=17, max_features=27,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=20, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=76,
                       n_jobs=None, oob_score=False, random_state=17, verbose=0,
                       warm_start=False)
```

LightGBM, используем multiclass

Ввод [17]:
```
model_lgb = lgb.LGBMRegressor(random_state=17, max_depth=18,
            min_child_samples=18, num_leaves=75, n_estimators=1000,
            objective="multiclass", num_class=8)
model_lgb.fit(x, data_train["Response"])
```

```
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\lightgbm\engine.py:148: UserWarning: Found
`num_boost_round` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argument".format(alias))
```

Out[17]:
```
LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
              importance_type='split', learning_rate=0.1, max_depth=18,
              min_child_samples=18, min_child_weight=0.001, min_split_gain=0.0,
              n_estimators=100, n_jobs=-1, num_boost_round=1000, num_class=8,
              num_leaves=75, objective='multiclass', random_state=17,
              reg_alpha=0.0, reg_lambda=0.0, silent=True, subsample=1.0,
              subsample_for_bin=200000, subsample_freq=0)
```

Расчет предсказаний

Кроме непосредственно значений дополнительно посчитаем вероятности совпадения с тем или иным классом

Ввод [18]:
```
x_test = pd.DataFrame(data_test, columns=columns_transformed)
```

Ввод [19]:
```
data_test_svm_proba = pd.DataFrame(model_svm.predict_proba(x_test))
```

Ввод [20]:
```
data_test_logr_proba = pd.DataFrame(model_logr.predict_proba(x_test))
```

Ввод [21]:
```
data_test_rf_proba = pd.DataFrame(model_rf.predict_proba(x_test))
```

Ввод [22]:
```
data_test_lgb = pd.DataFrame(model_lgb.predict(x_test))
```

Несколько вариантов ансамблей с голосованием (выбор класса выполняется для каждого кортежа по отдельности):

- "Мягкое" голосование (в том числе, с определенными весами): суммируются вероятности каждого класса среди всех оценок, выбирается наибольшее.
- "Жесткое" (мажоритарное) голосование: выбирается самый популярный класс среди моделей (число моделей должно быть нечетным).
- Отсечение: из вероятностей моделей выбирается только наиболее значимые, например, больше 0.3.
- Экспертное голосование: вес оценки эксперта зависит от кортежа данных и самого класса (например, если определенная модель предсказывает определенный класс точнее других).

Здесь используем "мягкое" голосование, для этого необходимо рассчитать вероятности всех класса для каждого кортежа данных.

Ввод [23]:
```
def vote_class (x):
    a = np.argmax(x.values)
    return a
```

Ввод [71]:
```
data_test_proba = data_test_svm_proba.copy()
for i in range(0, 8):
    data_test_proba[i] = 5*data_test_proba[i]
    data_test_proba[i] = data_test_proba[i] + 5*data_test_logr_proba[i]
    data_test_proba[i] = data_test_proba[i] + data_test_rf_proba[i]
    data_test_proba[i] = data_test_proba[i] + 12*data_test_lgb[i]
data_test_proba["target"] = data_test_proba.apply(vote_class, axis=1)
```

Оценка ансамбля

Рассчитаем оценку взвешенного предсказания 4 моделей

Кластеризация дает 0.192, kNN(100) - 0.382, лог. регрессия - 0.512/0.496, SVM - 0.95, реш. дерево - 0.3, случайный лес - 0.487, XGBoost - 0.536, градиентный бустинг - 0.56, LightGBM - 0.569, CatBoost - 0.542

Ввод [72]:
```
print ("Ансамбль классификации:",
       round(cohen_kappa_score(data_test_proba["target"],
                               data_test["Response"], weights="quadratic"), 3))
```

```
Ансамбль классификации: 0.56
```

Матрица неточностей

Ввод [73]:
```
print ("Ансамбль классификации\n",
    confusion_matrix(data_test_proba["target"], data_test["Response"]))
```

```
Ансамбль классификации
 [[ 300  151   15   27   53  123   31   18]
 [ 172  329   20    8   99   87   25   21]
 [  22   23   74   13    0    1    0    0]
 [  32   22   56  182    0    2    1    4]
 [  88  142   17    0  568   85    8    5]
 [ 259  266   19   21  227 1255  324  122]
 [ 103   96    1    5   46  262  606  123]
 [ 250  232    3   34  109  491  596 3603]]
```

Само-проверка

Проверим, насколько ансамбль хорошо предсказывает обучающие данные

Ввод [74]:
```
data_copy = data_train.copy()
x_copy = pd.DataFrame(data_copy, columns=columns_transformed)
```

Ввод [75]:
```
data_copy_svm = pd.DataFrame(model_svm.predict_proba(x_copy))
```

Ввод [76]:
```
data_copy_logr = pd.DataFrame(model_logr.predict_proba(x_copy))
```

Ввод [77]:
```
data_copy_rf = pd.DataFrame(model_rf.predict_proba(x_copy))
```

Ввод [78]:
```
data_copy_lgb = pd.DataFrame(model_lgb.predict(x_copy))
```

Ввод [79]:
```
for i in range(0, 8):
    data_copy_svm[i] = 5*data_copy_svm[i]
    data_copy_svm[i] = data_copy_svm[i] + 5*data_copy_logr[i]
    data_copy_svm[i] = data_copy_svm[i] + data_copy_rf[i]
    data_copy_svm[i] = data_copy_svm[i] + 12*data_copy_lgb[i]
target = data_copy_svm.apply(vote_class, axis=1)
```

Ввод [80]:
```
print ("Результат:",
       round(cohen_kappa_score(target,
                  data_copy["Response"], weights="quadratic"), 3))
```

```
Результат: 0.964
```

Ввод [82]:
```
print (confusion_matrix(target, data_copy["Response"]))
```

```
[[ 4741     0     0     0     0     0     0     0]
 [    1  5041     0     0     0     0     0     0]
 [    0     0   808     0     0     0     0     0]
 [    0     0     0  1138     0     0     0     0]
 [    3     6     0     0  4289     0     0     0]
 [   38    46     0     0     8  8476     6     2]
 [   13    13     0     0     1    15  5889     1]
 [  185   185     0     0    32   436   541 15590]]
```

РАСЧЕТ РЕЗУЛЬТАТОВ

Постановка задачи

Загрузим данные, приведем их к числовым, заполним пропуски, нормализуем данные и оптимизируем память.

Построим LightGBM модель с оптимальными параметрами. Выгрузим результаты расчетов в требуемом формате.

Данные:

1. https://video.ittensive.com/machine-learning/prudential/train.csv.gz
2. https://video.ittensive.com/machine-learning/prudential/test.csv.gz
3. https://video.ittensive.com/machine-learning/prudential/sample_submission.csv.gz

Подключение библиотек

Ввод [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, confusion_matrix
import lightgbm as lgb
from sklearn import preprocessing
```

Загрузка данных

Ввод [2]:
```
data = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/train.csv.gz")
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 128 entries, Id to Response
dtypes: float64(18), int64(109), object(1)
memory usage: 58.0+ MB
None
```

Предобработка данных

Ввод [3]:
```
def data_preprocess (df):
    df["Product_Info_2_1"] = df["Product_Info_2"].str.slice(0, 1)
    df["Product_Info_2_2"] = pd.to_numeric(df["Product_Info_2"].str.slice(1, 2))
    df.drop(labels=["Product_Info_2"], axis=1, inplace=True)
    for l in df["Product_Info_2_1"].unique():
        df["Product_Info_2_1" + l] = df["Product_Info_2_1"].isin([l]).astype("int8")
    df.drop(labels=["Product_Info_2_1"], axis=1, inplace=True)
    df.fillna(value=-1, inplace=True)
    data["Response"] = data["Response"] - 1
    return df
```

Ввод [4]:
```
data = data_preprocess(data)
```

Набор столбцов для расчета

Ввод [5]:
```
columns_groups = ["Insurance_History", "InsuredInfo", "Medical_Keyword",
                  "Family_Hist", "Medical_History", "Product_Info"]
columns = ["Wt", "Ht", "Ins_Age", "BMI"]
for cg in columns_groups:
    columns.extend(data.columns[data.columns.str.startswith(cg)])
print (columns)
```

```
['Wt', 'Ht', 'Ins_Age', 'BMI', 'Insurance_History_1', 'Insurance_History_2', 'Insurance_History_3', 'Insurance_History_4', 'Insurance_History_5', 'Insurance_History_7', 'Insurance_History_8', 'Insurance_History_9', 'Medical_Keyword_1', 'Medical_Keyword_2', 'Medical_Keyword_3', 'Medical_Keyword_4', 'Medical_Keyword_5', 'Medical_Keyword_6', 'Medical_Keyword_7', 'Medical_Keyword_8', 'Medical_Keyword_9', 'Medical_Keyword_10', 'Medical_Keyword_11', 'Medical_Keyword_12', 'Medical_Keyword_13', 'Medical_Keyword_14', 'Medical_Keyword_15', 'Medical_Keyword_16', 'Medical_Keyword_17', 'Medical_Keyword_18', 'Medical_Keyword_19', 'Medical_Keyword_20', 'Medical_Keyword_21', 'Medical_Keyword_22', 'Medical_Keyword_23', 'Medical_Keyword_24', 'Medical_Keyword_25', 'Medical_Keyword_26', 'Medical_Keyword_27', 'Medical_Keyword_28', 'Medical_Keyword_29', 'Medical_Keyword_30', 'Medical_Keyword_31', 'Medical_Keyword_32', 'Medical_Keyword_33', 'Medical_Keyword_34', 'Medical_Keyword_35', 'Medical_Keyword_36', 'Medical_Keyword_37', 'Medical_Keyword_38', 'Medical_Keyword_39', 'Medical_Keyword_40', 'Medical_Keyword_41', 'Medical_Keyword_42', 'Medical_Keyword_43', 'Medical_Keyword_44', 'Medical_Keyword_45', 'Medical_Keyword_46', 'Medical_Keyword_47', 'Medical_Keyword_48', 'Family_Hist_1', 'Family_Hist_2', 'Family_Hist_3', 'Family_Hist_4', 'Family_Hist_5', 'Medical_History_1', 'Medical_History_2', 'Medical_History_3', 'Medical_History_4', 'Medical_History_5', 'Medical_History_6', 'Medical_History_7', 'Medical_History_8', 'Medical_History_9', 'Medical_History_10', 'Medical_History_11', 'Medical_History_12', 'Medical_History_13', 'Medical_History_14', 'Medical_History_15', 'Medical_History_16', 'Medical_History_17', 'Medical_History_18', 'Medical_History_19', 'Medical_History_20', 'Medical_History_21', 'Medical_History_22', 'Medical_History_23', 'Medical_History_24', 'Medical_History_25', 'Medical_History_26', 'Medical_History_27', 'Medical_History_28', 'Medical_History_29', 'Medical_History_30', 'Medical_History_31', 'Medical_History_32', 'Medical_History_33', 'Medical_History_34', 'Medical_History_35', 'Medical_History_36', 'Medical_History_37', 'Medical_History_38', 'Medical_History_39', 'Medical_History_40', 'Medical_History_41', 'Product_Info_1', 'Product_Info_3', 'Product_Info_4', 'Product_Info_5', 'Product_Info_6', 'Product_Info_7', 'Product_Info_2_2', 'Product_Info_2_1D', 'Product_Info_2_1A', 'Product_Info_2_1E', 'Product_Info_2_1C', 'Product_Info_2_1B']
```

Нормализация данных

Ввод [6]:
```
scaler = preprocessing.StandardScaler()
data_transformed = pd.DataFrame(scaler.fit_transform(pd.DataFrame(data,
                                                                  columns=columns)))
columns_transformed = data_transformed.columns
data_transformed["Response"] = data["Response"]
```

Оптимизация памяти

Ввод [7]:
```
def reduce_mem_usage (df):
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if str(col_type)[:5] == "float":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.finfo("f2").min and c_max < np.finfo("f2").max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo("f4").min and c_max < np.finfo("f4").max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
        elif str(col_type)[:3] == "int":
            c_min = df[col].min()
            c_max = df[col].max()
            if c_min > np.iinfo("i1").min and c_max < np.iinfo("i1").max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo("i2").min and c_max < np.iinfo("i2").max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo("i4").min and c_max < np.iinfo("i4").max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo("i8").min and c_max < np.iinfo("i8").max:
                df[col] = df[col].astype(np.int64)
        else:
            df[col] = df[col].astype("category")
    end_mem = df.memory_usage().sum() / 1024**2
    print('Потребление памяти меньше на', round(start_mem - end_mem, 2), 'Мб (минус', round(100 * (start_mem - end_mem)
                                                                                            / start_mem, 1), '%)')
    return df
```

Ввод [8]:
```
data_transformed = reduce_mem_usage(data_transformed)
print (data_transformed.info())
```

```
Потребление памяти меньше на 40.49 Мб (минус 75.1 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59381 entries, 0 to 59380
Columns: 119 entries, 0 to Response
dtypes: float16(118), int8(1)
memory usage: 13.4 MB
None
```

LightGBM

Рассчитаем модель по оптимальным показателям. Возможно уточнение/дообучение уже на всей выборке без разбиения на обучающую/тестовую.

Ввод [17]:
```
x = pd.DataFrame(data_transformed, columns=columns_transformed)
model = lgb.LGBMRegressor(random_state=17, max_depth=17,
                          min_child_samples=18, num_leaves=35,
                          n_estimators=1000)
```

Построим модель

Ввод [18]:
```
model.fit(x, data["Response"])
```

Out[18]:
```
LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
              importance_type='split', learning_rate=0.1, max_depth=17,
              min_child_samples=18, min_child_weight=0.001, min_split_gain=0.0,
              n_estimators=1000, n_jobs=-1, num_leaves=35, objective=None,
              random_state=17, reg_alpha=0.0, reg_lambda=0.0, silent=True,
              subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

Загрузка данных для расчетов

Применим построенную модель для расчета актуальных данных.

Будем использовать ранее рассчитанные значения нормализация данных.

Ввод [26]:
```
data_test = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/test.csv.gz")
data_test = data_preprocess(data_test)
data_test = reduce_mem_usage(data_test)
data_test_transformed = pd.DataFrame(scaler.transform(pd.DataFrame(data_test,
                                                                   columns=columns)))
print (data_test_transformed.info())
```

```
Потребление памяти меньше на 16.34 Мб (минус 84.9 %)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19765 entries, 0 to 19764
Columns: 118 entries, 0 to 117
dtypes: float32(118)
memory usage: 8.9 MB
None
```

Предсказание данных и оценка модели

LightGBM возвращает дробное значение класса, его нужно округлить.

Дополнительно приведем значение класса к диапазону 1...8

Ввод [27]:
```
data_test_transformed["Response"] = np.round(model.predict(data_test_transformed)) + 1
data_test_transformed["Response"] = (
    data_test_transformed["Response"].apply(lambda x:1 if x<1 else 8 if x>8 else x)
)
print (data_test_transformed.head())
```

```
          0         1         2         3         4         5         6  \
0  0.519789  1.002921  1.045952  0.022140  0.611857 -0.169414 -1.159587
1  0.215425  0.266273  1.122714  0.126020 -1.634368 -0.169414  0.862391
2  0.308653  0.022915  0.894903  0.405695  0.611857 -0.169414 -1.159587
3 -0.278139 -0.707156  0.592804  0.143999 -1.634368 -0.169414  0.862391
4 -0.513953 -0.463799 -0.542540 -0.333447  0.611857 -0.169414 -1.159587

          7         8         9  ...       109       110       111       112  \
0  1.101046 -1.156735  1.130555  ... -0.083689  0.441621 -0.149284 -0.200031
1 -1.013721  0.864261 -0.928723  ... -0.083689  0.441621 -0.149284 -0.666860
2  1.101046 -1.156735  1.130555  ... -0.083689  0.441621 -0.149284 -0.200031
3 -1.013721  0.862242  0.100916  ... -0.083689 -2.264385 -0.149284 -1.133690
4  1.101046 -1.156735  1.130555  ... -0.083689  0.441621 -0.149284 -1.133690

        113       114       115       116       117  Response
0  0.750845 -0.623305 -0.216001 -0.128866 -0.142142       4.0
1 -1.331832  1.604350 -0.216001 -0.128866 -0.142142       6.0
2  0.750845 -0.623305 -0.216001 -0.128866 -0.142142       6.0
3 -1.331832  1.604350 -0.216001 -0.128866 -0.142142       7.0
4 -1.331832  1.604350 -0.216001 -0.128866 -0.142142       6.0

[5 rows x 119 columns]
```

Формирование результатов

Загрузим пример данных для отправки и заменим в нем столбец Response на рассчитанный ранее.

Ввод [28]:
```
submission = pd.read_csv("https://video.ittensive.com/machine-learning/prudential/sample_submission.csv.gz")
print (submission.head())
```

```
   Id  Response
0   1         8
1   3         8
2   4         8
3   9         8
4  12         8
```

Ввод [29]:
```
submission["Response"] = data_test_transformed["Response"].astype("int8")
print (submission.head())
```

```
   Id  Response
0   1         4
1   3         6
2   4         6
3   9         7
4  12         6
```

Выгрузка результатов

Ввод [30]:
```
submission.to_csv("submission.csv", index=False)
```

Часть 7
ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

Искусственные нейроны сети (или просто нейросети) являются некоторым отдельным классом методов машинного обучения и представляют собой целую огромную область изучения, потому что, фактически, вбирают в себя все остальные методы машинного обучения и преобразуют их, используя некоторые нелинейные функции, в частности функцию активации и нейроны, о которых далее пойдет речь.

Не нужно относиться к нейронным сетям как к еще одному методу, с помощью которого можно решить задачу. Скорее, нейронные сети – это некоторый класс методов. Разные методы в нейронных сетях имеют свою собственную архитектуру. Последовательно будем изучать все основные архитектуры, а также изучим как нейронную сеть можно использовать, как она может быть использована в задачах регрессии, классификации, сегментации, задачах обучения без учителя и задачах обучения с подкреплением.

Ещё один важный аспект, который нужно иметь в виду, прежде чем вы начнете погружаться в нейронные сети: нейронные сети обычно требуют некоторого количества оборудования, то есть то оборудование, которое у вас существует (локальные или серверные мощности), которое подходит для задач машинного обучения, может оказаться недостаточными для задач обучения нейронных сетей и особенно глубоких нейронных сетей. Поэтому будьте готовы к тому, чтобы использовать более мощное оборудование. Однако в целом, большинство задач сейчас решаются на современном оборудовании вполне нормально.

Сейчас в каждом мобильном телефоне есть свои нейронные сети (которые обрабатывают фотографии, украшают наши лица, размывают фон фотографии и так далее), поэтому можно с уверенностью сказать, что нейронные сети давно вошли в нашу жизнь.

Начало изучения нейронных сетей было положено в середине 20 века. С тех пор испытала несколько подъемов и спадов, сейчас достаточно уверенно они используются практически везде, где мы о них знаем, потому что позволяют с любой точностью аппроксимировать любую неизвестную функцию. Поскольку нейронная сеть – это одна большая нелинейная функция, то с ее помощью мы можем решить практически произвольную задачу. Понятно, что существует ряд тривиальных задач, которые не решаются с применением нейронных сетей, например, задача забивания гвоздя в стену. Но, если задача не решается с помощью нейронной сети, то, скорее всего, для неё [задачи] уже

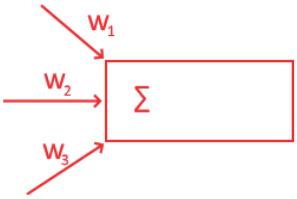
предложены другие методы машинного обучения, которые мы изучили или которые мы изучим. В целом, класс задач, которые хорошо решаются нейронными сетями, они, скорее всего, хорошо решаются только ими (это большой класс работы с текстами, изображениями, видео, где обычные, линейные методы машинного обучения не пригодны).

Изучение мира сети всегда начинается с каких-то аналогий. Естественно, прежде чем нейронные сети появились на свет, ученые попытались понять, как работает мозг человека и каким-то образом попытались его смоделировать. И выяснилось, что мозг состоит из нейронов (небольших клеток, каждый из которых имеет несколько десятков или даже сотен отростков – синапсов, которые связывают эту одну клетку с другими). Именно от этой картины и появилось понятие нейронной сети, то есть совокупность связанных нейронов между собой через синапсы. Схематично нейроны можно изобразить кружочками, а синапсы – линиями. Также у каждого есть основной синапс, который именуют аксоном.

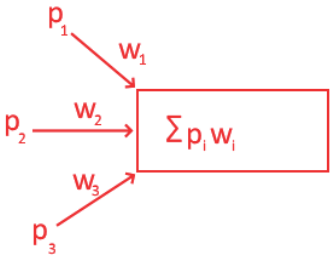
Таким образом, у нас имеется базовая модель нейронной сети, которую ученые взяли из биологии и адаптировали ее для задач машинного обучения. В тесно связанной сети, когда у нас присутствуют многие десятки сотен узлов, связанных друг с другом, сигнал в них начинает передаваться разными замысловатыми формами маршрутами.

Что нам еще нужно знать про нейрон, чтобы дальше приступить к освоению архитектуры и пониманию того, как это работает?

Понятно, что нейрон передает сигнал, однако, кроме «передаточного пункта», каждый нейрон представляет собой некоторую нелинейную функцию от весов. Если изобразить нейрон в виде черного ящика, то он может получать сигнал от одного входящего нейрона (может получать один или несколько входящих сигналов). Предполагают, что у нас по умолчанию этих сигналов на нейрон приходит несколько (в современных нейронных сетях – это десятки – сотни тысяч входящих сигналов). Эти сигналы приходят с какими-то весами (изобразим схематично веса как w_1, w_2 и w_3).



Первое, что нейрон выполняет – простое суммирование входных сигналов, помноженных на вес.



Умножение организовано потому, что нейрон не просто суммирует входящий поток, он оценивает входы. То есть у каждого входа нейрона есть свой вес и этим как раз и достигается первая часть нелинейности (поскольку у нас таких нейронов и весов очень много, мы можем за счет этого очень многообразно представить исследуемые явления). Второй нелинейностью (основной фактор, которым достигается нелинейность), которая позволяет аппроксимировать любую произвольную функцию с помощью нейронной сети, является функция активации (подробнее про функцию активации материал будет в следующей теме), она оперирует суммой весов и выдает некоторый сигнал. То есть, если у весов маленькая функция активации, она может ничего не выдать на вход, а просто сбросить напряжение; и также она передает дальше по сети этот собственно сигнал. У каждого нейрона есть свой вес на входящую связь и один и тот же сигнал, переданный от одного нейрона дальше, естественно, по-разному дойдет до следующих нейронов. Это реализовано именно на базе входящих весов.

Принято, что у нас есть входящие веса на нейрон и он ими оперирует, а выходящие веса (веса выходящих связей) относятся уже к следующему слою нейронной сети.

Итак, что нам нужно знать про нейрон. Нейрон – это некоторый черный ящик, некоторая нелинейная функция, которую оперирует сигналами, входящими в него от других нейронов. Если это, например, входной нейрон, то он просто получает некоторое число (то есть у него один единственный вход с весом 1 и это у нас условно просто ввод пользователя или распознавание цвета пикселя), это будет входной и тогда нейроны активируются этим значением. В общем случае, у нейрона не один вход и у этих входов есть некоторые ненулевые и не единичные веса (от 0 до 1). По этим входам у нас идут сигналы на нейрон, нейрон получает взвешенную сумму сигнал, от нее вычисляет некоторую функцию активации нелинейно и передает этот сигнал дальше.

СЛОИ В НЕЙРОСЕТЯХ

Мы разобрались как работает один единственный нейрон. Он получает на вход некоторые сигналы, суммирует эти сигналы с некоторыми весами, и затем по этой взвешенной сумме сигналов принимает решение и отправляет свой сигнал дальше. Понятно, что веса у нас лежат в диапазоне значений от 0 до 1, сигнал у нас тоже, скорее всего, нормирован от нуля до единицы и выходной сигнал нейрона тоже нормируют от 0 до 1, чтобы далее не работать с переполнением – это сугубо математические упрощения.

Давайте теперь посмотрим, как у нас нейроны объединяются в слои. Мы уже посмотрели, что у нас должен быть обязательно в нейронной сети некоторый входной слой (на самом деле это некоторый ряд значений, он совершенно условно называет эти нейроны некоторым входным слоем. Это просто ячейки памяти, куда мы помещаем первоначальные изменения). Фактически – это структура данных напоминает некоторый список или массив значений, потому что у этих нейронов один единственный вход с весом 1, и в этот один единственный вход мы кладем какие-то определенные значения. Просто для дальнейшей работы и удобства обращения с нейронной сетью инициализируют этот набор значений как входной слой (по умолчанию подразумевают, что входной слой значений с весом 1).

Входной слой нейронной сети с значением веса, равным единице:

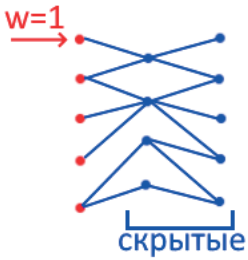
На вход нейронной сети мы подаем некоторые числа. Как и в любой модели машинного обучения, у нас есть вектор X, который мы подаем (набор параметров, которые мы подаем на вход) и есть некоторые ожидаемые значения, которые мы ожидаем на выходе (наши предсказываемые значения или классифицируемые значения).

Другие слои в нейронной сети называются, разумеется, по-другому, потому что они имеют уже некоторое другое назначение.

Дальше у нас может идти некоторое количество (их обычно достаточно много слоев, каким-то образом связанных с предыдущим слоем) слоёв, которые называются скрытыми.

Термин скрытости состоит в том, что к скрытым слоям нет прямого доступа. Мы подаем значения на вход входного слоя и скрытые слои изначально не видны (их можно вытащить, но в модели машинного обучения они не видны для того, кто пользуется этой моделью). Видны только входной слой (количе-

ство нейронов, которые мы должны запомнить) и виден выходной слой (например, в задачи классификации на выходном слое будет число нейронов, равных количеству классов).



На входной слой мы подаем параметры задачи, далее происходит некоторое вычисления в скрытых слоях (каким образом оно происходит, мы подробнее будем разбираться в следующих разделах) и на выходном слое у нас оказывается некоторое распределение по классам. Дальше производится обучение нейронной сети и, подавая входные данные, мы на выходе получаем какое-то значение, которое нейронная сеть.

Таким образом, существует три базовых слоя в любой нейронной сети. Их так условно разделили больше по признаку функционала, потому что на входной слой мы подаем наш набор параметров, от которого производятся вычисления. В выходном слое мы получаем ответ. И какие-то важные параметры для получения конечного ответа будут в скрытом слое.

Но в любом случае, у нас есть некоторый выходной слой, который мы каким-то образом должны обучать. Именно он будет содержать ответ нейронной сети на поставленную задачу.

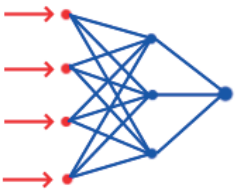
Есть некоторые скрытые слои каким-то образом друг с другом связанные. Отметим, что каждый нейрон получает на вход какие-то сигналы, суммирует их с определенными весами и выдает на выходе какое-то значение, причем на каждом выходе он выдает одно и то же значение. Одно и то же значение на входе разные нейроны понимают по-разному, потому что у них разные входные веса у этого сигнала. Собственно, благодаря этому достигается многообразие, которое позволяет решить произвольную нелинейную задачу.

НЕЙРОН СМЕЩЕНИЯ

Поговорив про основные типы нейронов (входящие нейроны, скрытые и нейроны выходящего слоя), мы должны обязательно упомянуть про такую абстракцию как нейроны смещения (bias – нейроны).

Зачем они нужны, какую они задачу решают и можем ли мы обойтись без них? Вспомним, что у нас есть некоторый входной слой (например, 4 эле-

мента), на него мы подаем некоторые значения, для которых требуется обработка нейросети. И у нас есть некоторое количество скрытых нейронов, например, мы на рисунке изобразим один слой полносвязной сети, и у нас есть, например, один выходной нейрон, куда все три нейрона так или иначе входят.



Однако, поскольку у нас на входе значение каждого нейрона формируется как вес всех входящих связей, у нас получается некоторая однозначная зависимость от входных данных, и мы не можем выходные данные сдвинуть относительно входных. Говоря иначе, уравнение, где y – это выход, у нас получится следующая формула:

$$y = w_1x_1 + w_2x_2 + w_3x_3,$$

где x_n – это значение n-ого нейрона в скрытом слое;
w_u – это соответствующие веса нейронов.

В таком случае, наш выход будет однозначно определен от всех трех нейронов. Но, например, у нас нет такой задачи, что у нас функция однозначно определена от входного вектора (у нее может быть некоторое среднее значение, которое вообще никак не зависит от входного вектора). Поэтому нейрон смещения утоняет предсказание модели.

Разумеется, что нейрон смещения может быть на каждом скрытом слое. И, естественно, поскольку у нас входные связи нормируются по весам, то значение, которое дает нейрон смещение одно и тоже. Его принимают условно равным единице, а как именно каждый нейрон его воспринимает на следующем слое, определяется весом, который определяется по методу обратного распространения ошибки (метод будет рассмотрен далее).

Нейрон смещения (или просто смещение) нужно для уточнения ситуации в том случае, когда наше предсказываемое значение не полностью определяется входными параметрами, а есть некоторый свободный член, который независим от входных параметров.

Это не самая частая задача в нейронных сетях, но иногда она имеет место быть. Естественно, нейроны смещения тогда обычно добавляются как отдельный гиперпараметр обучения нейронной сети. Однако, не всегда нейроны смещения являются «серебряной пулей», потому что в большинстве случаев, все смещения так или иначе можно воспроизвести с помощью дополнительных входов (используя уже имеющуюся информацию).

Итого, у нас есть 4 разных варианта для нейронов, это:

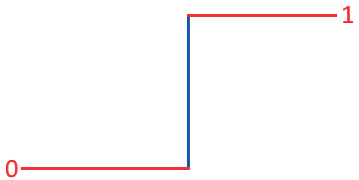
1) нейроны входного слоя;
2) нейроны скрытого слоя;
3) нейроны выходного слоя;
4) нейроны смещения (вес).

ФУНКЦИИ АКТИВАЦИИ

Функция активации является самой интересной частью нелинейности в нейронных сетях. Именно на них базируется действительная сложность во многом всей нейросети. Функции активации отвечают за быструю сходимость или несходимость нейронной сети, также они ответственны за правильное решение тех или иных задач обучения с учителем.

Давайте по порядку будем разбираться, что мы хотим от функции активации и как она в целом влияет на наш результат.

Рассмотрим задачу бинарной классификации. Задача нейронной сети – определить 0 или 1 на выходе системы при некоторых входных условиях. У нас есть два порога (0 и 1 как два возможных, крайних значения) и, в зависимости от какого-то характерного порога, будет происходить срабатывание функции активации. Тогда функция активации напоминала бы ступеньку, соединяющую пороги по нормали – это идеальная функция активации:



Математически эта функция решается через $sign(x)$ по аргументу x. В общем виде функция выглядит следующим образом:

$$F(x) = \frac{1 + sign(x)}{2}.$$

Когда x отрицателен, $sign(x)$ принимает значение минус один. Соответственно, функция принимает значение, равное 0, когда $x < 0$. И $F(x) = 1$, когда $x > 0$.

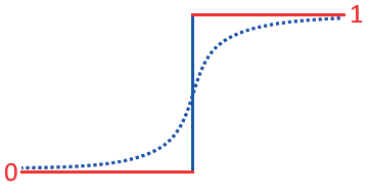
Однако, в практическом понимании алгоритмизации – это бесполезная функция активации, потому что она не обучаемая.

При решении мы либо промахиваемся, либо не промахиваемся. Одновременно с этим (что уже напоминает «колесо фортуны»), мы также не знаем, насколько мы промахиваемся, потому что переход функции (выделено красном на предыдущем снимке) – это абсолютно ровная линия, отсутствует малейший изгиб.

В практическом смысле, нейрон, имеющий такую функцию активации, не обучаем.

Поэтому ученые решили найти какие-то очень похожие функции, но которые можно было бы обучать.

Первым, наиболее изученным, подходом является сигмоида:



$$y = \frac{1}{1+e^{-x}}.$$

Аналитически задается следующим образом:

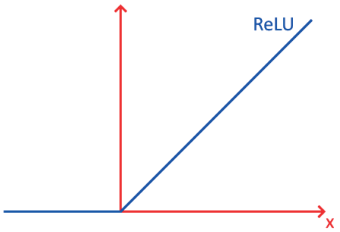
$$F(x) = \frac{1}{1+e^{-x}}.$$

Из рисунка очевидно, что сигмоида идет не горизонтально, у нее есть некоторый наклон, который резко набирает силу и тоже уходит практически на асимптоту (ориентировочно в районе значения 1 – 2).

Однако, выяснилось, что сигмоида – это не самый лучший вариант, поскольку у нас появляются двойные вычисления (кроме того, что нужно посчитать значение функции, ещё нужно посчитать степень экспоненты). Это дополнительное усложнение вычислений на каждом шаге обучения нейронной сети. Когда у нас немного нейронов (100, 1000, 10000) это не так много для текущих процессорных мощностей, но когда речь идет про миллионы (десятки миллионов) нейронов, то, к сожалению, каждые 10–15 % дополнительного времени обучения – это существенно (прирост в несколько суток / месяцев).

Поэтому сигмоида – очень хорошая функция активации, но она обычно используется в основном на финальном слое в задачах бинарной классификации.

Во внутренних слоях используется полулинейная функция ReLU (от английского Rectified Linear Unit). Ее график выглядит гораздо проще:



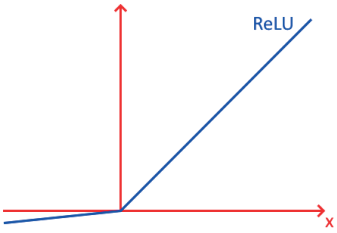
До 0 по OX она по своему значению равна 0, а вторая часть находится под углом к оси OX, чтобы можно было обучать систему.

С некоторой точностью она повторяет сигмоиду, но ReLU вычислительно проще. Поэтому на внутренних слоях нейронной сети обычно в качестве функции активации используют ReLU.

В реальности на практике скорость ее сходимости (за счет того, что она вычисляется быстрее) просто выше. По эпохам нейронных сетей она возможно немного проигрывает сигмоиде (именно в плане параметра скорости обучения), однако, за счет того, что эпохи в ReLU обучаются быстрее, то она, естественно, вырывается вперед.

У ReLU есть одна неприятная особенность: если у нас изначально веса инициализированы таким образом, что получаются значения меньше нуля, то тогда ReLU у нас всегда выдает 0. То есть, если входной сигнал оказывается меньше нуля, то для полулинейной функции всегда будут получены функции, равные 0. Это значит, что на своей первой части (до значения $x = 0$, то есть при всех $x < 0$) ReLU не обучается. Эти нейроны называются «мертвыми» нейронами ReLU. В среднем, половина нейронов у такой нейронной сети (у которой полулинейная функция активации) не работают.

Что с этим можно сделать? Можно нарастить нейронную сеть в два раза (пометить пакеты, прошедшие мимо и обучить сеть вновь). Другим подходом избежания проблемы является «исправленная полулинейная функция активации».



Поскольку движение вдоль оси OX невозможно, то можно прибегнуть к ухищрению и немного наклонить функцию к оси OX в отрицательной области значений оси OX.

Полулинейная функция используется как базовая в задачах регрессии. Сигмоида используется для бинарной классификации (когда два класса). ReLU применяется для задачи регрессии, когда нам нужно получить некоторое значение (оно домножается со сдвигом и достаточно активно влияет на метод обратного распространения ошибки – чуть более используемый, чем сигмоида вариант).

Ещё одной важной разновидностью функции активации является SoftMax. Она используется в задачах мультиклассовой классификации (множественной классификации). Аналитически представляется возведением в степень входного сигнала (p*w):

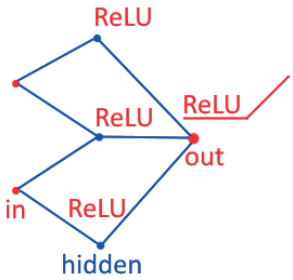
$$F = \frac{e_k^{p_i w_i}}{\sum e_k^{p_i w_i}}.$$

На выходе таким образом мы получаем несколько нейронов, сумма всех выходов нейронов равна 1. Соответственно, мы получаем вероятность того, что наши входы принадлежат к одному из выходов (реальных классов). То есть, функция SoftMax – это просто математическое преобразование сигмоиды (взвешенная сигмоида) для случая множественной классификации. Не следует путать SoftMax с принципиально новой математической моделью.

ОБРАТНОЕ РАСПРОСТРАНЕНИЕ ОШИБКИ

Давайте теперь рассмотрим метод обратного распространения ошибки, который является одним из фундаментальных концептов, парадигмой мира нейронных сетей и позволил иметь то машинное обучение, которое мы сейчас имеем.

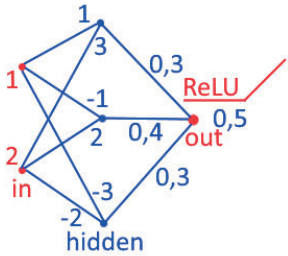
В середине 20 века архитектура нейронных сетей зашла в тупик и не смогли быть промышленным образом использованы, потому что было непонятно, что дальше делать с предсказанной ошибкой нейронной сети. После введения метода обратного распространения ошибки инженерные и научные мысли двинулась дальше. Рассмотрим на конкретном примере как работает метод и что он позволяет делать.



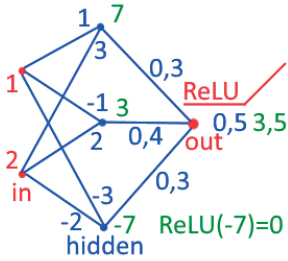
В качестве примера рассмотрим приведенную на картинке выше схему нейронной сети. В качестве функции активации везде используется ReLU. У функции ReLU производная равна значению, так как на положительном участке оси OX, она равна X.

Поэтому, когда мы сейчас будем рассматривать обратное распространение ошибки, у нас все будет достаточно просто с этой функцией. Именно поэтому она так сильно используется в нейронных сетях, потому что производная от нее (то, что позволяет нам посчитать ошибку – градиентный спуск) можно сказать даже не вычисляется, потому что это уже константа.

Например, у нас есть на входе значения 1 и 2, а на выходе требуется получить 0,5. Расставим соответствующие веса нейронной сети:



Давайте теперь посчитаем, что у нас в итоге получается. Вычисление вперед – это прямое распространение, то есть следование по принципу работы нейронной сети (слева направо, от входа к выходу).



В прямом распространении получаем ответ в 3,5. Вы можете самостоятельно умножить входные значения на вес, а затем просуммировать приходящие в нейрон значения. Однако, отметим тот нюанс, что отрицательные значения при подходе к выходу зануляются (умирание нейронов).

Теперь, получив значение больше, чем нужно, уменьшаем его. Для этого находим величину ошибки дельта:

$$\Delta = 0,5 - 3,5 = -3.$$

Далее нам необходимо умножить все веса в нейронной сети на это значение (поскольку мы промахнулись на него).

Затем вычисляются уменьшение весов для всех входящих в данный слой связей, после этого считается ошибка (то есть уменьшение веса на значения в этом нейроне). Возьмем средний отрезок, в нем уменьшение веса будет равно дельта (−3), умноженное на 0,4 (−1,2).

Далее мы поступаем аналогичным образом, только в каждом следующем возвратном шаге умножаем на ***предыдущее*** значение ошибки. Таким образом, ошибка как бы распространяется обратно, переигрывая все существующие установленные веса.

И на следующий итерации, имея те же самые значения на входе, мы будем иметь уже ***скорректированные*** веса.

Условно, на следующей итерации конечное значение станет приблизительно равным 2,7–2,8 (вместо 3,5). Разумеется, это более близкое значение к 0,5.

Таким образом за счет нескольких эпох (подробнее о понятии эпохи в следующих разделах) мы получим оптимальные веса на всех уровнях между всеми слоями нейронной сети. И эти веса будут давать нужный результат через некоторое количество эпох, что в конечном счете, приведет к точному решению задачи регрессии.

МНОГОСЛОЙНЫЙ ПЕРЦЕПТРОН

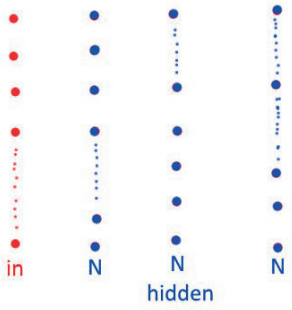
Многослойный перцептрон является первой и одной из наиболее простых архитектур нейронных сетей, с которых обычно начинается работа. Его можно в какой-то мере сравнить с моделью машинного обучения, такой как линейная регрессия (хоть и линейная регрессия и не совсем модель, а скорее некоторая математическая система уравнений, которую мы накладываем на наши входные данные).

Многослойный перцептрон уверенно работает с большим количеством входных параметров (понятно, что это не в любых масштабах. Например, для изображений, где у речь идет о десятках миллионов входных значений пикселей, многослойный перцептрон работает очень плохо).

Скрытых слоев может быть достаточно много в многослойном перцептроне. Однако, существует несколько базовых архитектур. Поэтому возможно некоторое количество одинаковых нейронов на скрытых слоях и скрытых слоев тоже некоторое количество (2, 3, 4, 5 и так далее). Аналогию можно провести с полиномиальной регрессией (один скрытый слой – это некоторая полулинейная или почти линейная аппроксимация – линейная регрессия).

Соответственно, количество слоев может быть по аналогии определено порядком полиномиальной регрессии.

Общий вид многослойного перцептрона в графическом представлении может быть изображен в следующем виде:

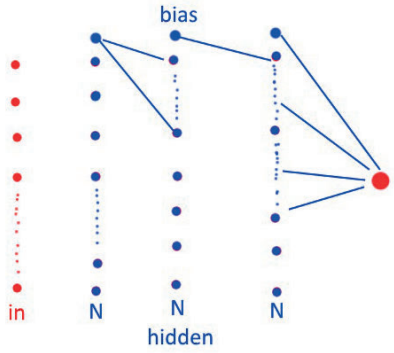


При задаче регрессии на выходе будет один выходной слой (так как нам требуется получить итоговое значение).

На каждом скрытом слое может быть нейрон смещения, необходимый для того, чтобы добиваться нужного смещения для выходного значения.

Всего у нас 4 типа нейронов на многослойном перцептроне:

1) входные;
2) нейроны скрытых слоев;
3) выходные;
4) нейроны смещения (bias).



Также, одной из чуть менее распространенных структур многослойного перцептрона является пирамидка. Мы говорим, что у нас есть некоторое число входных скрытых нейронов на первом скрытом слое, затем это количество уменьшается и мы пытаемся за счет умножения нейронов (перемножения связей между ними) выделить некоторые характерные особенности из наших данных. Схематично это можно изобразить следующим образом:

У нас количество связей с каждым скрытым слоем уменьшается. Это чуть менее распространенная архитектура, но в целом ничто не мешает исключить один или даже два скрытых слоя, заменив все небольшим количеством сужающих нейронов, когда мы через них пытаемся выделить основные параметры из всего многообразия факторов и их каким-то образом суммировать. Для задачи регрессии используется либо полулинейная функция активации, либо сигмоида.

Такая архитектура также позволяет учесть все многообразие факторов и высветить ровно те из них, которые могут повлиять на принятие решений.

В том случае, если у нас задача бинарной классификации, то у нас на выходном слое уже 2 нейрона (2 каких-то класса). По факту, это наша номинативная случайная величина, которая принимает два каких-то значения (их условно обозначаем 0 и 1). В качестве функции активации – сигмоида.

В случае, когда работа происходит с множественной классификацией, то выходной слой будет состоять, соответственно, из того числа нейронов, количество которых нужно нам по числу классов. Функция активации – SoftMax.

ЧАСТЬ ПРАКТИЧЕСКИХ НАВЫКОВ К 7

ЗАДАЧА ПРЕДСКАЗАНИЯ ФОРМЫ ОБЛАКОВ

Постановка задачи

Даны изображения облаков, сделанные со спутника. Часть из этих изображений уже размечена учеными на содержание облаков определенной формы – это «цветок», «рыба», «сахар» и «гравий». Для оставшихся фотографий необходимо определить как класс области облаков, так и найти границу этой определенной области.

Загрузим данные уменьшенных изображений и проведем исследовательский анализ данных для них. Найдем все взаимосвязи, которые помогут в построении модели классификации.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz

Подключение библиотек

Ввод [1]:

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
from imgaug.augmentables.segmaps import SegmentationMapsOnImage
import requests
from PIL import Image
from io import BytesIO
```

Загрузка обучающих данных

Формат данных: название изображения+формы облаков, кодированная маска облаков

Кодирование маски выполняется в RLE: пары значений содержат начало - порядковый номер пиксела в изображении - и длину - число пикселей после начала, которые нужно включить в маску. Например, кодировка 1 3 10 5 - 1-начало, 3-длина, 10-начало, 5-длина, будет означать последовательность пикселей 1, 2, 3, 10, 11, 12, 13, 14, 15.

Пиксели нумеруются из левого верхнего угла сверху вниз слева направо. Первый пиксел имеет координаты (1,1), второй пиксел имеет координаты (1,2) и т.д.

Ввод [3]:

```
train = pd.read_csv("https://video.ittensive.com/machine-learning/clouds/train.csv.gz")
print (train.info())
print (train.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22184 entries, 0 to 22183
Data columns (total 2 columns):
Image_Label      22184 non-null object
EncodedPixels    11836 non-null object
dtypes: object(2)
memory usage: 346.7+ KB
None
          Image_Label                                      EncodedPixels
0    0011165.jpg_Fish  264918 937 266318 937 267718 937 269118 937 27...
1  0011165.jpg_Flower  1355565 1002 1356965 1002 1358365 1002 1359765...
2  0011165.jpg_Gravel                                               NaN
3   0011165.jpg_Sugar                                               NaN
4    002be4f.jpg_Fish  233813 878 235213 878 236613 878 238010 881 23...
```

Очистка данных

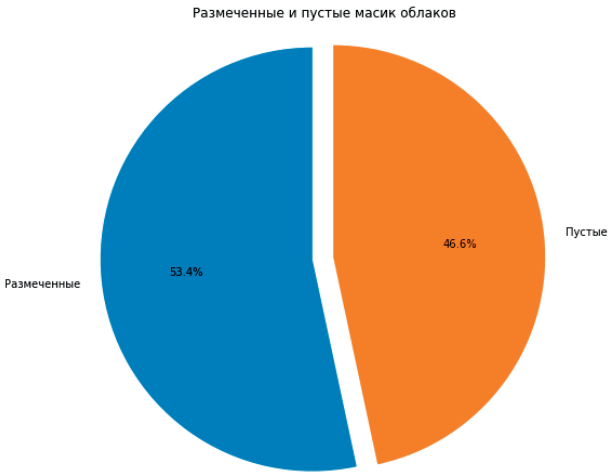
Разделим файлы и названия облаков на изображениях

Ввод [4]:
```
train["Image"] = train["Image_Label"].str.split("_").str[0]
train["Label"] = train["Image_Label"].str.split("_").str[1]
train.drop(labels=["Image_Label"], axis=1, inplace=True)
print (train.head())
```

```
                                         EncodedPixels        Image   Label
0  264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg    Fish
1  1355565 1002 1356965 1002 1358365 1002 1359765...  0011165.jpg  Flower
2                                                NaN  0011165.jpg  Gravel
3                                                NaN  0011165.jpg   Sugar
4  233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg    Fish
```

Размеченные данные

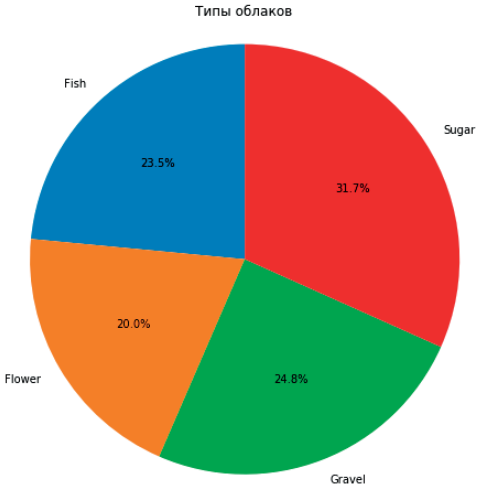
Ввод [5]:
```
sizes = [train.EncodedPixels.count(),
        len(train) - train.EncodedPixels.count()]
explode = (0, 0.1)
fig, ax = plt.subplots(figsize=(16,8))
ax.pie(sizes, explode=explode, labels=["Размеченные", "Пустые"],
       autopct="%1.1f%%", startangle=90)
ax.axis("equal")
ax.set_title("Размеченные и пустые масик облаков")
plt.show()
```



Формы облаков

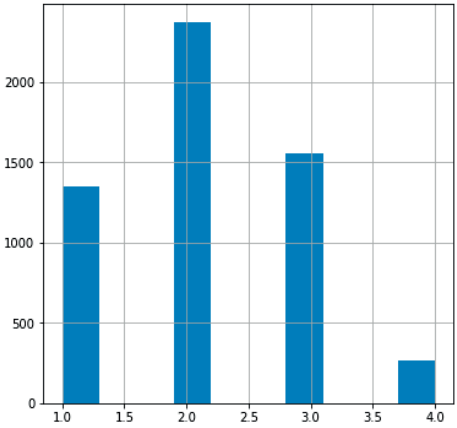
Ввод [6]:
```
fish = train[train["Label"] == "Fish"].EncodedPixels.count()
flower = train[train["Label"] == "Flower"].EncodedPixels.count()
gravel = train[train["Label"] == "Gravel"].EncodedPixels.count()
sugar = train[train["Label"] == "Sugar"].EncodedPixels.count()
sizes = [fish, flower, gravel, sugar]
```

Ввод [7]:
```
fig, ax = plt.subplots(figsize=(16,8))
ax.pie(sizes, labels=["Fish", "Flower", "Gravel", "Sugar"],
       autopct="%1.1f%%", startangle=90)
ax.axis("equal")
ax.set_title("Типы облаков")
plt.show()
```



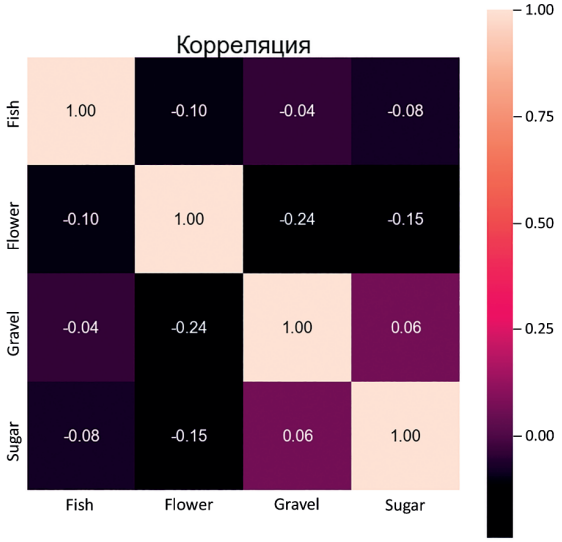
Количество разных облаков на изображениях

Ввод [8]:
```
_, area = plt.subplots(figsize=(6,6))
train.groupby("Image")['EncodedPixels'].count().hist(ax=area)
ax.set_title("Число разных облаков")
plt.show()
```



Корреляция между формами облаков

Ввод [9]:
```
labels = train["Label"].unique()
for label in labels:
    train["Label_" + label] = ((train["EncodedPixels"].notnull())
                               & (train["Label"] == label)).astype("int8")
train_corr = train.groupby("Image")["Label_Fish", "Label_Flower",
                                    "Label_Gravel", "Label_Sugar"].sum()
corrs = np.corrcoef(train_corr.values.T)
sns.set(rc={'font.size':20, 'figure.figsize': (12,12)})
sns.heatmap(corrs, cbar=True, annot=True, square=True, fmt='.2f',
            yticklabels=labels,
            xticklabels=labels).set_title("Корреляция", fontsize=30)
plt.show()
```



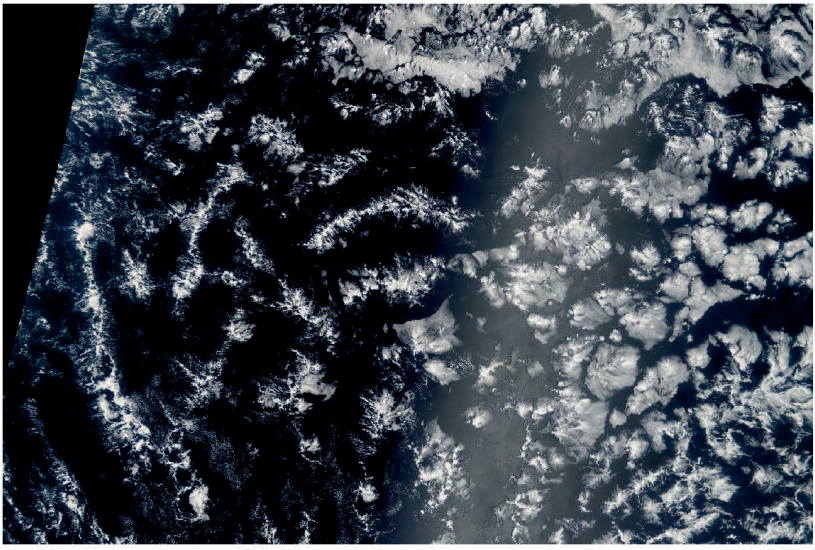
Области облаков на изображении

Отметим области, в которых присутствуют облака выделенного типа

Ввод [ ]:
```
image_x = 2100
image_y = 1400
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")
def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(" ")]).reshape(-1,2)
    mask = np.zeros(x*y, dtype="uint8")
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l + i, x, y)] = 1
    return mask.reshape(y,x).transpose()
```

Загрузим изображение и преобразуем его в PNG для вывода через matplotlib

Ввод [13]:
```
img = requests.get("https://video.ittensive.com/machine-learning/clouds/train_images/" +
                   train["Image"].values[0])
img = Image.open(BytesIO(img.content))
image_png = BytesIO()
img.save(image_png, format="PNG")
image_png.seek(0)
plt.figure(figsize=(21,14))
plt.axis("off")
plt.imshow(mpimg.imread(image_png))
```



Ввод [16]:
```
img = np.array(Image.open(image_png))
mask = calc_mask(train["EncodedPixels"].values[0])
segmap = SegmentationMapsOnImage(mask, mask.shape)
fig, area = plt.subplots(figsize=(21,14))
area.axis("off")
plt.title("Fish")
area.imshow(np.array(segmap.draw_on_image(img)).reshape(img.shape))
```

ПРЕДОБРАБОТКА ИЗОБРАЖЕНИЙ

Постановка задачи

Загрузим данные изображений, построим на них размеченные области, сформируем массив серых пикселей из изображений.

Выгрузим массив пикселей в HDF5 для дальнейшей работы.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz

Подключение библиотек

Ввод [1]:
```
import numpy as np
import pandas as pd
from PIL import Image
```

Загрузка данных

Ввод [3]:
```
train = pd.read_csv("https://video.ittensive.com/machine-learning/clouds/train.csv.gz")
print (train.head())
```

```
          Image_Label                                        EncodedPixels
0    0011165.jpg_Fish  264918 937 266318 937 267718 937 269118 937 27...
1  0011165.jpg_Flower  1355565 1002 1356965 1002 1358365 1002 1359765...
2  0011165.jpg_Gravel                                                NaN
3   0011165.jpg_Sugar                                                NaN
4    002be4f.jpg_Fish  233813 878 235213 878 236613 878 238010 881 23...
```

Очистка данных

Отделим категории и область каждой формы облаков

Ввод [4]:
```
train["Image"] = train["Image_Label"].str.split("_").str[0]
train["Label"] = train["Image_Label"].str.split("_").str[1]
train.drop(labels=["Image_Label"], axis=1, inplace=True)
print (train.head(20))
```

```
                                         EncodedPixels        Image   Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg    Fish
1   1355565 1002 1356965 1002 1358365 1002 1359765...  0011165.jpg  Flower
2                                                  NaN  0011165.jpg  Gravel
3                                                  NaN  0011165.jpg   Sugar
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg    Fish
5   1339279 519 1340679 519 1342079 519 1343479 51...  002be4f.jpg  Flower
6                                                  NaN  002be4f.jpg  Gravel
7   67495 350 68895 350 70295 350 71695 350 73095 ...  002be4f.jpg   Sugar
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg    Fish
9   2047 703 3447 703 4847 703 6247 703 7647 703 9...  0031ae9.jpg  Flower
10                                                 NaN  0031ae9.jpg  Gravel
11  658170 388 659570 388 660970 388 662370 388 66...  0031ae9.jpg   Sugar
12                                                 NaN  0035239.jpg    Fish
13  100812 462 102212 462 103612 462 105012 462 10...  0035239.jpg  Flower
14  65400 380 66800 380 68200 380 69600 380 71000 ...  0035239.jpg  Gravel
15                                                 NaN  0035239.jpg   Sugar
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg    Fish
17                                                 NaN  003994e.jpg  Flower
18  353317 416 354717 416 356117 416 357517 416 35...  003994e.jpg  Gravel
19  28011 489 29411 489 30811 489 32211 489 33611 ...  003994e.jpg   Sugar
```

Ввод [5]:
```
data = pd.DataFrame({"Image": train["Image"].unique()})
for l in train["Label"].unique():
    data[l] = pd.Series(train[train["Label"] == l]["EncodedPixels"].values)
print (data.head())
```

```
          Image                                               Fish  \
0  0011165.jpg  264918 937 266318 937 267718 937 269118 937 27...
1  002be4f.jpg  233813 878 235213 878 236613 878 238010 881 23...
2  0031ae9.jpg  3510 690 4910 690 6310 690 7710 690 9110 690 1...
3  0035239.jpg                                                NaN
4  003994e.jpg  2367966 18 2367985 2 2367993 8 2368002 62 2369...

                                              Flower  \
0  1355565 1002 1356965 1002 1358365 1002 1359765...
1  1339279 519 1340679 519 1342079 519 1343479 51...
2  2047 703 3447 703 4847 703 6247 703 7647 703 9...
3  100812 462 102212 462 103612 462 105012 462 10...
4                                                NaN
```

```
                                                 Gravel  \
0                                                   NaN
1                                                   NaN
2                                                   NaN
3  65400 380 66800 380 68200 380 69600 380 71000 ...
4  353317 416 354717 416 356117 416 357517 416 35...

                                                  Sugar
0                                                   NaN
1  67495 350 68895 350 70295 350 71695 350 73095 ...
2  658170 388 659570 388 660970 388 662370 388 66...
3                                                   NaN
4  28011 489 29411 489 30811 489 32211 489 33611 ...
```

Обработка изображений

Предварительно загрузим весь архив с изображениями и распакуем его в train_images_small

Пример: 0011165.jpg, размеры 525 * 350 = 183750 пикселей

Каждое изображение приведем к серой палитре, результат загрузим в фрейм данных.

Ввод [6]:
```
imgdata = np.array([np.zeros(183750, dtype="uint8")]*len(data))
for i, img in enumerate(data["Image"].unique()):
    imgdata[i] = np.array(Image.open("train_images_small/" +
                 img).convert("L"), dtype="uint8").reshape(1, -1)[0]
imgdata = pd.DataFrame(imgdata)
print (imgdata.head())
```

```
   0       1       2       3       4       5       6       7       8       \
0       0       0       0       0       0       0       0       0       0
1      71      42      31      38      48      68      70      46      24
2      97      98     100     102     105     107     109     110     111
3      87      86      89      93      93      88      86      86      81
4      18      19      21      21      20      19      17      16      14

   9       ...  183740  183741  183742  183743  183744  183745  183746  \
0       0  ...      82      65      52      70      64     102     113
1      21  ...      98     113     110      59      27      38      57
2     112  ...       0       0       0       0       0       0       0
3      81  ...     181     187     170     149     145     165     164
4      15  ...      66      75     112     156     140     125     137

   183747  183748  183749
0     102      96     116
1      82     110     133
2       0       0       0
3     108      70      72
4      92      59      48

[5 rows x 183750 columns]
```

Ввод [7]:
```
for column in data.columns:
    imgdata[column] = data[column]
del data
print (imgdata.head())
```

```
    0   1    2    3    4    5    6    7    8    9  ...  183745  183746  \
0   0   0    0    0    0    0    0    0    0    0  ...     102     113
1  71  42   31   38   48   68   70   46   24   21  ...      38      57
2  97  98  100  102  105  107  109  110  111  112  ...       0       0
3  87  86   89   93   93   88   86   86   81   81  ...     165     164
4  18  19   21   21   20   19   17   16   14   15  ...     125     137

   183747  183748  183749        Image  \
0     102      96     116  0011165.jpg
1      82     110     133  002be4f.jpg
2       0       0       0  0031ae9.jpg
3     108      70      72  0035239.jpg
4      92      59      48  003994e.jpg

                                                Fish  \
0  264918 937 266318 937 267718 937 269118 937 27...
1  233813 878 235213 878 236613 878 238010 881 23...
2  3510 690 4910 690 6310 690 7710 690 9110 690 1...
3                                                NaN
4  2367966 18 2367985 2 2367993 8 2368002 62 2369...

                                               Flower  \
0  1355565 1002 1356965 1002 1358365 1002 1359765...
1  1339279 519 1340679 519 1342079 519 1343479 51...
2  2047 703 3447 703 4847 703 6247 703 7647 703 9...
3  100812 462 102212 462 103612 462 105012 462 10...
4                                                NaN

                                               Gravel  \
0                                                NaN
1                                                NaN
2                                                NaN
3  65400 380 66800 380 68200 380 69600 380 71000 ...
4  353317 416 354717 416 356117 416 357517 416 35...

                                                Sugar
0                                                NaN
1  67495 350 68895 350 70295 350 71695 350 73095 ...
2  658170 388 659570 388 660970 388 662370 388 66...
3                                                NaN
4  28011 489 29411 489 30811 489 32211 489 33611 ...

[5 rows x 183755 columns]
```

Сохраняем данные в HDF5

Потребуется до 3 Гб оперативной памяти

Ввод [8]:
```
imgdata.to_hdf("clouds.data.h5", "data", format="fixed",
               compression="gzip", complevel=9, mode="w")
```

```
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\pandas\io\pytables.py:274: PerformanceWarni
ng:
your performance may suffer as PyTables will pickle object types that it cannot
map directly to c-types [inferred_type->mixed-integer,key->axis0] [items->None]

  f(store)
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\pandas\core\generic.py:2377: PerformanceWar
ning:
your performance may suffer as PyTables will pickle object types that it cannot
map directly to c-types [inferred_type->mixed,key->block0_values] [items->['Image', 'Fish', 'Flower', 'Gravel', 'Suga
r']]

  return pytables.to_hdf(path_or_buf, key, self, **kwargs)
```

ОПОРНЫЕ ВЕКТОРЫ И КОЭФФИЦИЕНТ СХОДСТВА

Постановка задачи

Загрузим подготовленные данные из HDF5. Разделим данные на обучающие и проверочные и построим модель опорных векторов для типа облака Fish.

Проведем оценку качества предсказания по F1 и коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/clouds.data.h5 (959 Мб)

Подключение библиотек

Ввод [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import f1_score
```

Загрузка данных

Ввод [2]:
```
clouds = pd.read_hdf("clouds.data.h5")
print (clouds.head())
```

```
   0   1   2   3   4   5   6   7   8   9  ...  183745  183746  \
0   0   0   0   0   0   0   0   0   0   0  ...     102     113
1  71  42  31  38  48  68  70  46  24  21  ...      38      57
2  97  98 100 102 105 107 109 110 111 112  ...       0       0
3  87  86  89  93  93  88  86  86  81  81  ...     165     164
4  18  19  21  21  20  19  17  16  14  15  ...     125     137

   183747  183748  183749        Image  \
0     102      96     116  0011165.jpg
1      82     110     133  002be4f.jpg
2       0       0       0  0031ae9.jpg
3     108      70      72  0035239.jpg
4      92      59      48  003994e.jpg

                                                Fish  \
0  264918 937 266318 937 267718 937 269118 937 27...
1  233813 878 235213 878 236613 878 238010 881 23...
2  3510 690 4910 690 6310 690 7710 690 9110 690 1...
3                                                NaN
```

Оставим только данные по Fish, в примере мы только на них будем обучать модель

Ввод [3]:
```
clouds.drop(labels=["Image", "Flower", "Gravel", "Sugar"],
            axis=1, inplace=True)
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [4]:
```
clouds_train, clouds_test = train_test_split(clouds, test_size=0.2)
clouds_train = pd.DataFrame(clouds_train)
clouds_test = pd.DataFrame(clouds_test)
del clouds
print (clouds_train.head())
```

```
        0    1   2   3   4   5   6   7   8   9  ...  183741  183742  183743  \
3163   22   22  22  22  22  22  22  22  22  22  ...      24      21      25
5476   66   66  57  58  79  90  77  64  74  48  ...     167     174     177
3203  147  105  60  36  17  32  59  39  39  43  ...      57      63      61
2198   41   53  43  31  41  48  45  49  67  91  ...     189     184     182
3555   87   89  91  93  94  94  93  93  90  94  ...      55      56      59

      183744  183745  183746  183747  183748  183749  \
3163      42      76     148     176     163     199
5476     201     186     185     173     154     151
3203      58      69      80     142     199     201
2198     176     157     151     171     182     159
3555      62      66      65      64      63      63

                                                  Fish
3163  791043 817 792443 817 793843 817 795243 817 79...
5476  294857 530 296257 530 297657 530 299057 530 30...
3203                                                NaN
2198                                                NaN
3555                                                NaN

[5 rows x 183751 columns]
```

Метод опорных векторов

Последовательно рассчитываем коэффициенты для пакетов по 100 изображений, иначе может не поместиться в оперативную память.

Используем warm_start=True, чтобы переиспользовать предыдущие параметры.

Ввод [5]:
```
y = clouds_train["Fish"].notnull().astype("int8")
x = pd.DataFrame(clouds_train).drop(labels=["Fish"], axis=1)
model = SGDClassifier(loss="log", warm_start=True)
```

Ввод [6]:
```
for i in range (len(clouds_train)//100):
    model.partial_fit(x[i*100:i*100+100], y[i*100:i*100+100], classes=[0,1])
```

Ввод [7]:
```
del x
del y
```

Средняя область облаков

В качестве локализации облаков на изображении возьмем среднюю область по обучающей выборке

Ввод [8]:
```
image_x = 2100
image_x_4 = image_x//4
image_y = 1400
image_y_4 = image_y//4
```

Ввод [9]:
```
def locate_rectangle (a):
    vals = [int(i) for i in a[0].split(" ")]
    x = vals[0]//image_y
    y = vals[0]%image_y
    width = (image_x + (vals[-2] + vals[-1])//image_y - x)%image_x
    height = (vals[-2] + vals[-1])%image_y - y
    return [x, y, width, height]
```

Ввод [10]:
```
areas = pd.DataFrame(clouds_train["Fish"].copy().dropna(axis=0))
areas = areas.apply(locate_rectangle, axis=1, result_type="expand")
coords = np.array(areas.mean()//4)
print (coords)
```

```
[ 94. 103. 332. 111.]
```

Ввод [12]:
```
sgd_mask = np.zeros(image_x_4*image_y_4,
                    dtype="uint8").reshape(image_x_4, image_y_4)
for x in range(image_x_4):
    for y in range(image_y_4):
```

```
        if (x >= coords[0] and x<= (coords[0] + coords[3]) and
           y >= coords[1] and y<= (coords[1] + coords[3])):
            sgd_mask[x][y] = 1
sgd_mask = sgd_mask.reshape(image_x_4*image_y_4)
print (sgd_mask.sum())
```

```
12544
```

Предсказание значений

Ввод [19]:
```
result = pd.DataFrame({"EncodedPixels":clouds_test["Fish"],
                       "Is_Fish": clouds_test["Fish"].notnull().astype("int8")})
result["target"] = model.predict(pd.DataFrame(clouds_test).drop(labels=["Fish"],
                                                                axis=1))
print (result.head(10))
```

```
                                              EncodedPixels  Is_Fish  target
2447  21087 398 22487 398 23887 398 25287 398 26687 ...         1        1
2192  31262 337 32662 337 34062 337 35462 337 36862 ...         1        0
1741                                                NaN         0        0
4343  189766 312 191166 312 192566 312 193966 312 19...         1        0
2628                                                NaN         0        0
855                                                 NaN         0        1
3693  519 458 1919 458 3319 458 4719 458 6119 458 75...         1        1
1826  47607 546 49007 546 50407 546 51807 546 53207 ...         1        1
1632  2208618 530 2210018 530 2211418 530 2212818 53...         1        0
1092  777 492 2177 492 3577 492 4977 492 6377 492 77...         1        1
```

Оценка точности предсказания: F1

Точность = TruePositive / (TruePositive + FalsePositive)

Полнота = TruePositive / (TruePositive + FalseNegative)

F1 = 2 * Полнота * Точность / (Полнота + Точность)

Оценка точности предсказания: F1

Точность = TruePositive / (TruePositive + FalsePositive)

Полнота = TruePositive / (TruePositive + FalseNegative)

F1 = 2 * Полнота * Точность / (Полнота + Точность)

Ввод [16]:
```
print ("Опорные векторы:",
       round(f1_score(result["Is_Fish"], result["target"]), 3))
print ("Все Fish:",
       round(f1_score(result["Is_Fish"], np.ones(len(result))), 3))
```

```
Опорные векторы: 0.512
Все Fish: 0.671
```

Оценка по Дайсу

Для каждого изображения и каждой фигуры считается пересечение площади обнаруженной фигуры (X) с ее реальной площадью (Y) по формуле:

$$Dice = \frac{2 * |X \cap Y|}{|X| + |Y|}$$

Если и X, и Y равны 0, то оценка принимается равной 1. Оценка берется как среднее по всем фигурам.

Пока будем считать, что при определении типа облака на изображении, оно целиком размещено на фотографии: т.е. область облака - это все изображение.

Дополнительно посчитаем точность предсказания, что на фотографиях вообще нет облаков нужного типа.

Ввод [23]:
```
image_x = 525
image_y = 350
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(" ")]).reshape(-1, 2)
    mask = np.zeros(y*x, dtype="uint8")
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l + i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
#       target = sgd_mask
        target = np.ones(image_x*image_y, dtype="uint8")
        dice = 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice
```

Ввод [24]:
```
dice = result.apply(calc_dice, axis=1, result_type="expand")
print ("Опорные векторы, Fish:", round(dice.mean(), 3))
print ("Нет облаков, Fish:",
       round(len(result[result["Is_Fish"]==0])/len(result), 3))
```

```
Опорные векторы, Fish: 0.337
Нет облаков, Fish: 0.495
```

ДВУХСЛОЙНЫЙ ПЕРЦЕПТРОН

Постановка задачи

Загрузим подготовленные данные из HDF5. Разделим данные на обучающие и проверочные и построим двухслойный перцептрон для типа облака Fish.

Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/clouds.data.h5 (959 Мб)

Подключение библиотек

Ввод [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
```

Используемые функции

Ввод [2]:
```
image_x = 525
image_y = 350
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice = 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice
```

Загрузка данных

Ввод [3]:
```
clouds = pd.read_hdf('clouds.data.h5')
print (clouds.head())
```

```
    0   1    2    3    4    5    6    7    8    9  ...  183745  183746  \
0   0   0    0    0    0    0    0    0    0    0  ...     102     113
1  71  42   31   38   48   68   70   46   24   21  ...      38      57
2  97  98  100  102  105  107  109  110  111  112  ...       0       0
3  87  86   89   93   93   88   86   86   81   81  ...     165     164
4  18  19   21   21   20   19   17   16   14   15  ...     125     137

   183747  183748  183749        Image  \
0     102      96     116  0011165.jpg
1      82     110     133  002be4f.jpg
2       0       0       0  0031ae9.jpg
3     108      70      72  0035239.jpg
4      92      59      48  003994e.jpg

                                                Fish  \
0  264918 937 266318 937 267718 937 269118 937 27...
1  233813 878 235213 878 236613 878 238010 881 23...
2  3510 690 4910 690 6310 690 7710 690 9110 690 1...
3                                                NaN
```

Оставим только данные по Fish, только на них в данном примере будем обучать модель машинного обучения

Ввод [4]:
```
clouds.drop(labels=["Image", "Flower", "Gravel", "Sugar"],
    axis=1, inplace=True)
```

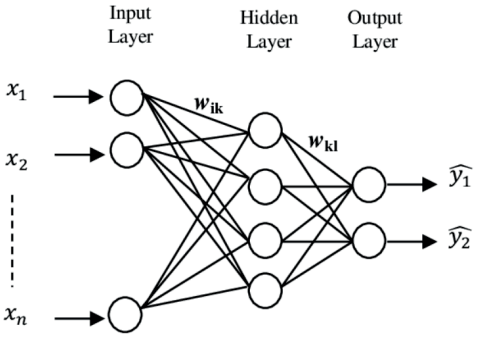
Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [5]:
```
clouds_train, clouds_test = train_test_split(clouds, test_size=0.2)
clouds_train = pd.DataFrame(clouds_train)
clouds_test = pd.DataFrame(clouds_test)
del clouds
print (clouds_train.head())
```

```
        0    1    2    3    4    5    6    7    8    9  ...  183741  183742  \
1841   42   32   32   25    4    0    2    2    0    0  ...      19      18
5196  149  149  148  147  147  146  146  145  146  145  ...      90      92
2006  104  134  114   56   30   34   39   43   50   77  ...      46      49
1948   39   36   32   30   31   32   33   33   32   32  ...     220     227
4561  161  159  145  163  180  165  153  143  142  142  ...     161     214

      183743  183744  183745  183746  183747  183748  183749  \
1841      17      17      17      17      17      17      18
5196      95      92      98     118     119     102      97
2006      43      43      44      45      45      46      47
1948     225     227     220     218     211     212     226
4561     236     241     228     235     219     197     202

                                                      Fish
1841  1768642 682 1770042 682 1771442 682 1772842 68...
5196                                                 NaN
2006  15839 323 17034 597 18434 597 19834 597 21234 ...
1948                                                 NaN
4561  1345165 235 1346565 235 1347965 235 1349365 23...

[5 rows x 183751 columns]
```

Двухслойный перцептрон



Последовательно рассчитываем коэффициенты для пакетов по 100 изображений, иначе может не поместиться в оперативную память.

Используем warm_start=True, чтобы переиспользовать предыдущие параметры.

Ввод [7]:
```
y = clouds_train["Fish"].notnull().astype("int8")
x = pd.DataFrame(clouds_train).drop(labels=["Fish"], axis=1)
model = MLPClassifier(hidden_layer_sizes=(31,),
                      max_iter=20, activation="logistic",
                      verbose=10, random_state=1, learning_rate_init=.02,
                      warm_start=True)
```

Ввод [12]:
```
for i in range(len(clouds_train)//100):
    model.partial_fit(x[i:i+100], y[i:i+100], classes=[0, 1])
```

```
Iteration 1, loss = 0.72517492
Iteration 2, loss = 0.69139934
Iteration 3, loss = 0.69218115
Iteration 4, loss = 0.68995657
Iteration 5, loss = 0.69197143
Iteration 6, loss = 0.69507116
Iteration 7, loss = 0.69910045
Iteration 8, loss = 0.69878797
Iteration 9, loss = 0.69840872
Iteration 10, loss = 0.70072479
Iteration 11, loss = 0.70232555
Iteration 12, loss = 0.70359634
Iteration 13, loss = 0.70379597
Iteration 14, loss = 0.70544680
Iteration 15, loss = 0.70782095
```

```
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 16, loss = 0.70839137
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 17, loss = 0.70856360
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 18, loss = 0.70944267
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 19, loss = 0.70964819
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 20, loss = 0.70895729
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 21, loss = 0.71133075
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 22, loss = 0.71188986
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 23, loss = 0.71224260
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 24, loss = 0.71448954
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 25, loss = 0.71446970
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 26, loss = 0.71322945
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 27, loss = 0.71348998
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 28, loss = 0.71048460
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 29, loss = 0.71189176
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 30, loss = 0.71161251
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 31, loss = 0.71410100
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 32, loss = 0.71414996
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 33, loss = 0.71141037
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 34, loss = 0.71302719
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 35, loss = 0.71665775
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 36, loss = 0.71366820
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 37, loss = 0.71583937
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 38, loss = 0.71380622
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 39, loss = 0.71372494
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 40, loss = 0.71127399
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 41, loss = 0.70911602
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 42, loss = 0.70797922
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 43, loss = 0.71149940
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Iteration 44, loss = 0.70895588
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

Ввод [13]:
```
del x
del y
```

Предсказание значений

Ввод [14]:
```
result = pd.DataFrame({"EncodedPixels": clouds_test["Fish"]})
result["target"] = model.predict(clouds_test.drop(labels=["Fish"],
                                                  axis=1))
print (result.head(10))
```

```
                                            EncodedPixels  target
2545                                                  NaN       0
588   1663220 690 1664620 690 1666020 690 1667420 69...       0
1852                                                  NaN       0
5237  1332363 411 1333763 411 1335163 411 1336563 41...       0
927   9477 314 10877 314 12277 314 13677 314 15077 3...       0
5096  6604 394 8004 394 9404 394 10804 394 12204 394...       0
4846  2073 400 3473 393 3870 3 4873 390 5264 9 6273 ...       0
3162  280828 366 282228 366 283628 366 285028 366 28...       1
1840  373209 10 373223 4 373230 3 373234 2 373237 1 ...       0
3340  579631 833 581031 833 582431 833 583831 833 58...       0
```

Оценка по Дайсу

Пока будем считать, что при определении типа облака на изображении, оно целиком размещено на фотографии: т.е. область облака - это все изображение.

Нет облаков - 0.5, опорные векторы - 0.3

Ввод [15]:
```
dice = result.apply(calc_dice, axis=1, result_type="expand")
print ("MLP, Fish:", round(dice.mean(), 3))
```

```
MLP, Fish: 0.498
```

Часть 8
ОБУЧЕНИЕ НЕЙРОСЕТИ

ЭПОХИ, ПАКЕТЫ, ИТЕРАЦИИ

Начиная разговор про обучение нейронной сети, первоначально изучим некоторые особенности этого обучения.

Предыдущие рассмотренные алгоритмы машинного обучения в основном использовали итерации (некоторую последовательность) в обучении, однако, практически всегда это было очень быстро. Модель обучалась за кратчайшее время (секунды, минуты, максимум – часы), поэтому про эпохи, итерации и пакеты обычно не говорят.

Нейронная сеть как модель машинного обучения обычно сложна, и одна единственная эпоха обучения (обучение всех нейронов) на тысяче экземпляров может занимать существенное время (минуты, десятки минут, иногда даже часы), в то время как нам может потребоваться тысяча таких эпох обучения.

Эпоха – это прохождение сигнала в нейронной сети для всех данных:

За одну эпоху все исходные данные (которые обычно дробятся на пакеты) проходят небольшими пакетами через обучение и дают нам некоторую ошибку. Разумеется, что любая модель машинного обучения на любой итерации дает ошибку (некоторые модели дают маленькую ошибку, поэтому мы их можем уже использовать в Production для предсказаний).

Для того, чтобы нейронная сеть обучилась, обычно требуется не менее 50 эпох (минимум для простых задач). Однако, это число разумно только для тех случаев, когда у нас все очевидно с параметрами, а наша задача – подбор соответствующих весов, чтобы достаточно быстро приблизиться к какому-то результату.

Число эпох может достигать и десятки тысяч в сложных задачах.

На практике обозначены два основных подхода к обучению:

1. Менее распространен. По одному экземпляру наших данных проводим в прямом и обратном порядке нейронной сети. Затем считаем ошибки, обновляем веса. И так делаем с каждым экземпляром данных. Понятно, что такой подход немного увеличивает итоговую ошибку, так как происходит идеальный подгон точности под каждый результат.

2. Также существует пакетное обучение, когда по сети проходит по N экземпляров данных. Однако, следует помнить, что такое обучение ограничено размером оперативной памяти, поэтому всё очень быстро упирается в аппаратную составляющую.

Пакет (batch) – это N экземпляров из данных. Например, если у нас есть тысяча экземпляров данных, то мы можем разбить их на 200 пакетов, в каждом из которых будет по 5 экземпляров.

Итерация – это относительный синоним пакета. Итерации равны числу пакетов данных. То есть, если принять во внимание предыдущий пример, то при 1000 экземпляров будет 200 итераций, в каждой итерации пакет из 5 экземпляров данных.

Итерацию можно вычислить по следующей формуле:

$$\text{Итерация} = \frac{\text{Число данных}}{\textit{batch size}}.$$

Это три основных термина, которые используются в обучение нейронных сетей.

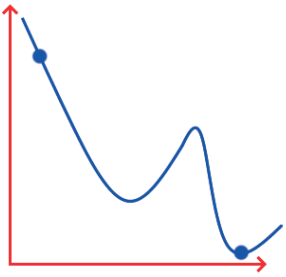
ОПТИМИЗАЦИЯ НЕЙРОННОЙ СЕТИ ПО НЕСТЕРОВУ

Поскольку даже простые базовые архитектуры нейросети состоят из большого числа параметров (обычно это тысячи – десятки тысяч параметров), то конечный вид функции, которую нам нужно оптимизировать, чтобы привести нейронную сеть к обученному состоянию, очень вычислительно сложен.

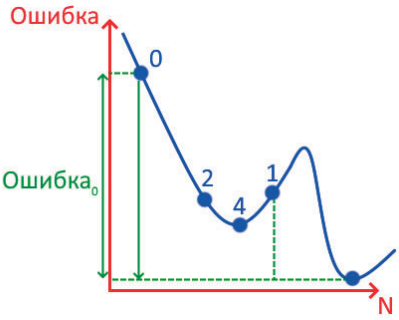
Оптимизировать функцию напрямую нельзя, то есть перед нами появляется невозможность аналитического решения задачи.

Также нельзя посчитать и вторую производную по градиенту, чтобы определить численно или хотя бы полу-аналитически решить проблему обучения.

Рассмотрим проблему обучения на классическом примере: у нас есть некоторая функция, для которой нам нужно найти минимум:



Проходя последовательно каждую точку (двигаясь по алгоритму градиентного спуска), однажды мы получим некоторый минимум:



Локальный минимум может оказаться больше глобального. Для решения этой проблемы можно прибегнуть к множественному поиску минимума при помощи ансамблирования нейронных сетей для поиска локального минимума.

Далее, усреднив значения локальных минимумов мы очень близко приблизимся к глобальному минимуму. В этом смысле (приближения к глобальному минимуму, а не он сам) нет какой-то принципиальной проблемы, потому что в целом нам требуется получить достаточно хорошее поведение обучения нейронной сети в плохих условиях (потому что реальные данные – это почти всегда плохие условия).

Способ градиентного спуска логичен, так как в его «инструкции» следует двигаться по уменьшению величины ошибки.

Стохастический градиентный спуск (случайный спуск) говорит нам о том, что следует двигаться не строго по весу ошибки, а по технологии выборки ряда элементов и уже их менять по величине ошибки, а остальные элементы не менять. Считается, что таким образом мы пытаемся промахнуться мимо локального минимума и выбиться в глобальный минимум. Хоть на практике это работает не очень хорошо, но как альтернатива градиентному спуску – вполне уверенный вариант.

Далее было решено изобрести какой-то новый метод для оптимизации функции. Вся проблема состоит в том, что неизвестна вторая производная ошибки. Первая производная показывает нам направление изменения, а вторая говорит про характер этого изменения (выпуклая или вогнутая функция).

Благодаря ней можно было посчитать «скорость скорости ошибки» и понять точки перегиба функции. Это позволило бы перейти через локальный минимум и искать глобальный дальше.

Первым подходом стал метод Нестерова. Метод пытается каким-то образом использовать идею второй производной, но он использует её не по всему ландшафту ошибки, а по ландшафту изменения наших параметров. Благодаря

этому мы узнаем ландшафт изменения веса параметра, например, с первой на вторую итерацию. Соответственно, зная это изменение, мы можем спрогнозировать некоторое поведение в целом. Далее открывается возможность посчитать вторую производную по уже известному движению. В целом, это работает достаточно хорошо, однако, сложно получить уверенные гарантии локального и глобального минимума.

Каким образом работать с поиском локально-глобального минимума или хотя бы избежать зацикленности функции на каком-то минимуме, будет рассмотрено в следующих разделах.

АДАПТИВНАЯ ОПТИМИЗАЦИЯ НЕЙРОННОЙ СЕТИ

Когда речь идет про обучение нейронной сети, то подразумевается, что существует достаточно много эпох обучения. Для того, чтобы решить задачу, мы подгоняем функцию ошибки к самому минимальному значению. Но «подогнать» функцию к минимуму (решить задачу минимизации функции ошибки) – это всего лишь термин, за которым скрывается подбор гиперпараметров (весов) в скрытых слоях нейронной сети.

Логично было бы предположить, что, если бы мы накапливали всю историю апробаций конкретных весов в каком-то отдельном месте (структуре памяти – массиве), то мы могли бы двигаться быстрее по процессу обучения нейросети. На практике же, это очень ресурсозатратно, поскольку даже для весов, исчисляющихся вторым порядком ($\approx 10^2$), то понадобится примерно столько же гигабайт оперативной памяти. Аппаратная оперативная память персонального компьютера очень дорога в цене (серверная ещё дороже по отношению ко времени ее потребления), поэтому возникает неразрешимая проблема потребления памяти.

Для решения проблемы была предложена другая оптимизация для ускорения работы нейронной сети. Необходимо работать с экспоненциальным средним, вместо работы с историей всех весов.

Экспоненциальное среднее рассчитывается следующим образом:

$$\Delta w = \Delta w_j * \frac{3}{4} + \frac{1}{4} * \Delta w_{j+1}.$$

Зная, как поменялся вес на предыдущем шаге (берем его за основу) и к этому изменению веса прибавляем некоторый нормировочный множитель (3/4 или ¼) на текущем шаге. Это мы считаем экспоненциальным средним – в значительной степени учитываем предыдущую историю и добавляем к предыдущей истории немного новую историю (обращаем внимание, что нормировочный множитель в сумме равен 1).

Альтернативным подходом к методу Нестерова является не накопление изменений вес, а попытка избежать осцилляций при обучении.

Эта идея получила название адаптивности или адаптивного градиента (сокращённо ***Ada***ptive ***Grad***ient – ***Adagrad***). Название подразумевает под собой набор методов обучения нейронной сети, когда нормирование происходит по сумме весов:

$$\Delta w = \Delta w_0 * \frac{1}{\sqrt{\sum_{x=1}^{N} \Delta w_x + \varepsilon}}.$$

Рассчитываем нормировку для каждой конкретной связи (в зависимости от истории изменения этой связи).

Смысл адаптивного градиента состоит в том, что нам меньше необходимо изменять часто изменяемые веса нейронов, так как они отвечают за некоторые характерные элементы обучения. Соответственно, адаптивный градиент позволяет сфокусироваться на нехарактерных элементах, что позволяет перескочить локальный минимум. Кроме этого, Adagrad позволяет обучать нейросеть на нехарактерных элементах, чтобы точнее обучить нейронную сеть.

Благодаря накоплению взвешенного среднего, экономятся и ресурсы потребляемой памяти. Высчитывается история изменения весов конкретного нейрона, чтобы можно было отследить как вес меняется со временем.

RMSPROP, ADADELTA, ADAM

Логичным продолжением идеи Adagrad является RMSprop – Root Mean Square Propogation. Переводя дословно «распространение по корню квадратному».

Идея RMSprop основана на Adagrad. Забегая немного вперед, отметим, что Adadelta, в свою очередь, придумана на RMSprop.

В основе Adagrad лежит идея нормирования по частоте весов. RMSprop нормирует не на частоту смены весов, а на корень квадратный нормировки:

$$\Delta p_i = \frac{\Delta p_0}{\sqrt{\sum_{x=1}^{n} \Delta * \delta_{ix}^2 + \varepsilon}}.$$

Вычисляя ошибку предсказания (по ней вычисляем историю ошибки предсказания по нашей связи – то есть меняя связь, как меняется ошибка предсказания по методу обратного распространения ошибки). Разумеется, используем экспоненциальное среднее ошибки предсказания, чтобы не нагружать всю оперативную память.

Основанные на Adadelta Adam и Adamax – это случай, когда выбираем дельту достаточно малой, чтобы на начальном этапе перескочить за локальный минимум и выйти в глобальной.

Однако, из-за неизвестного ландшафта функции ошибки, все оптимизационные методологии работают не так хорошо, как про них сказано. Поэтому единственное, что можно проверить на практике – это определить какая из методологий работает лучше всего.

RMSprop оперирует накопленной ошибкой по связи; Adadelta понадобится, чтобы RMSprop не занулял «хорошие» связи (то есть, не замораживать уже хорошо обученные связи, а сосредоточиться на плохо обученных в нейронной сети). Для того, чтобы сохранить хорошие связи, вводится корректирующий множитель:

$$\Delta p_i' = \Delta p_i * \sqrt{\sum_{x=1}^{N} \Delta p_{ix}}.$$

С его помощью мы проясняем, что, если ошибок мало, то мы можем обучать дальше. Если ошибка большая, то мы его зануляем связь, поскольку существует уверенное предположение в том, что он только раскачивает обучение и ничего хорошего не превозносит.

Изучим различия методологий от предыдущей формулы. Adamax оперирует тем, что берет корень не второй степени, а максимально возможной (например, корень 5 степени из p_{ix}^5). Это добавляет вычислений, но позволяет ещё немного лучше взглянуть на процесс обучения.

Подытожим вкратце, чему необходимо следовать для разрешения оптимизационных методов:

1) градиентный спуск (использует направление движения в сторону уменьшения ошибки на основе второй производной по функции);
2) стабилизация процесса обучения (следим за изменением параметров весов и за тем, чтобы они менялись не слишком резко – для этого стараемся их пенализируем);
3) быстрый поиск глобального минимума.

В совокупности методы дают очень хорошие результаты, позволяя снизить не только уровень ошибки, но и количество эпох обучения в каждом конкретном случае.

ОПТИМИЗАЦИЯ НЕЙРОННЫХ СЕТЕЙ

Разобрав все основные подходы к оптимизации нейронной сети, давайте теперь соберем их в одну группу и посмотрим, как устроен процесс оптимизации нейросетей.

Первое – это выбор архитектуры нейронной сети. Необходимо определиться с тем, что мы будем использовать в качестве основной методологии решения (алгоритм). Это могут быть сверточные, рекуррентные, реверсивные или другие нейронные сети, а также их модификации.

Если не очень понятно заранее, какую задачу предстоит решать, то к выбору архитектуры нейронной сети вполне нормально возвращаться не один раз. Причина очень проста – архитектура является основой и на нее накладывается все остальное.

Второй шаг. На архитектуру накладывают обычно пакетную нормализацию. Пакетную нормализацию обычно вставляют каждые два слоя (примерно после каждого полукаскада или сверточного каскада; также достаточно часто после каждого полносвязного слоя). Однако, количество слоев пакетной нормализации будет зависеть от некоторой (уже экспертной оценки), потому как данные получаем уже в процессе обучения и по ним проверяем помогла ли пакетная оптимизация или нет.

Третий шаг. Подключается регуляризация весов и отсев. Регуляризация – это нормировка весов на одном слое нейронной сети. Выполняется для того, чтобы сумма их модулей или сумма их квадратов весов была не очень большой, то есть коэффициент регуляризации подбирается экспертным подходом. Отметим, что к методам регуляризации относится и отсев. Слоев отсева обычно добавляют не очень много, потому что сильное разряжение за счет отсева нейронной сети не сильно поможет при ее обучении.

Регуляризацию добавляют на все (или на большинство) слои, чтобы максимально сгладить перегибы в обучении и усилить обобщающую силу. Зачастую отсев добавляют обычно после полносвязных слоев (каскада свертки).

Четвертый шаг. Кроме регуляризации на сверточных слоях также добавляют свою инициализацию весов, в зависимости от функций активации на фоне того эффекта, который мы хотим получить.

Существует несколько схем инициализации весов. Разумеется, схемы не случайны, во многом они зависят от параметров (тип распределения и так далее).

Пятый шаг. Выбор функции оптимизации. Наилучшими кандидатами являются Adam, Adamax, RMSprop. Когда совершенно ничего не будет подходить, то остается надежный вариант с градиентным бустингом. Он сведёт все параметры к наилучшим, но потребует несравненно большее время работы и требований к аппаратным ресурсам.

Большая скорость обучения не гарантирует того, что нейронная сеть сойдется быстрее. Малая скорость обучения более вероятен сход в минимум функцию ошибки.

Оставшийся шаг. Размер пакета обучения. Характеризует параметр, который можно представить в том, сколько экземпляров данных будет принято для обработки метода обратного распространения ошибки. Обычно его выбирают как некоторый множитель на число классов (если один класс, то будет достаточно 3–5 экземпляров надо собирать для обратного распространения ошибки).

Ориентировочно можно считать по следующей формуле, где множитель от 2 до 10 подбирается индивидуально к каждой задаче:

$$\text{Размер}_{\text{пакета}} = \text{Число}_{\text{классов}} * [2;10].$$

Желательно совершить как можно больше итераций для всех классов (каждого отдельно). Как можно больше – это то, насколько позволяют ваши ресурсы. Разумеется, что можно было применить и полный перебор, но чаще всего это невыгодно как по времени, так и невозможно по аппаратным ресурсам.

В целом, подход из совокупности данных этапов практически гарантирует вам, что вы решите задачу на основе нейронной сети наилучшим образом (при условии, что задача в принципе возможна для решения нейронной сетью).

Даже если задача решается плохо на нейросетях, то в любом случае вы найдете наилучшее хорошее решение из всех плохих решений.

ПАКЕТНАЯ НОРМАЛИЗАЦИЯ

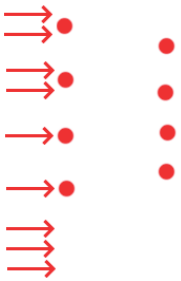
Обучение нейронной сети – это достаточно сложная функция, зависящая от многих параметров, в целом, неоднозначно.

Разрабатывались и продолжают разрабатываться различные подходы как это обучение сгладить и привести к интерпретируемому ограниченному виду, чтобы мы могли им управлять, чтобы скорость схождения нейронной сети к конечному результату была быстрее и так далее.

Одним из таких методов влияния на то, как нейронная сеть будет обучаться, является пакетная нормализация.

Давайте на небольшом примере разберем, как работает пакетная нормализация и зачем она вообще нужна. Чтобы понять как пакетная нормализация помогает, рассматривают проблему переобучения (так называемый ковариантный сдвиг).

У нас есть некоторый входной слой у нейронной сети, есть выходной слой:

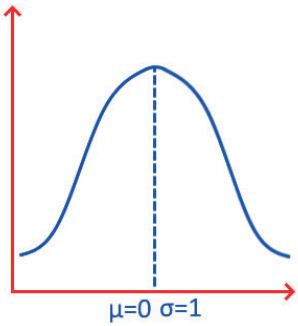


Допустим, нейронная сеть обучена на одних данных (только на количестве из определённого числа стрелок, например 5 или 6). Далее, если мы подадим теперь на вход нейронной сети три стрелки, то с ними нейронная сеть уже не сможет разобраться (даже, если они одного цвета, параметра и так далее. Изменено только количество).

Для борьбы с переобучением в этом направлении (достаточно однозначно определенных данных), предлагают сделать нормальную (Z-нормализацию) ***нормализацию значений***.

Пакетная нормализация (от английского «Batch Normalization» – «BN») отвечает за нормализацию входных значений. Нормализация проводится по каждому параметру в отдельности (при миллионе параметров для каждого из миллиона параметров по отдельности проходит пакет нормализации).

Пакетная нормализация предполагает математическую нормализацию. Графически данные выглядят следующим образом:



Параметр математического ожидания $\mu = 0$, а среднеквадратичное отклонение (дисперсия) $\sigma = 1$.

Входящие данные на каждом входном слое мы можем приводить к стандартным, то есть запоминая каждый раз преобразование, чтобы, обучившись, нейронная сеть могла воспроизвести это на тестовой выборке. Разумеется, что значения никак не теряются, так как мы не преобразовываем данные в пустоту. Параметры запоминаются и таким образом мы их можем использовать.

Для того, чтобы улучшить ситуацию, вводят дополнительное преобразование, которое включает в себя BN — это Афинное преобразование над нормализованным x. Нормализованный x это $\hat{x}$.

$$y = \gamma\hat{x} + \beta.$$

Гамма и бэта для каждого входного параметра фигурируют по отдельности. И эти параметры позволяют обратить наше преобразование (если оно было плохим нейронная сеть автоматически со временем сдвинет преобразование и вернется к исходным данным). Поскольку Аффинные преобразования лучше всего описывают ситуацию в машинном обучении.

Преобразования позволяют привести данные к некоторому стандартному виду, а потом применяем некоторые растяжения на данных и некоторые смещения.

Растяжение или сужение очевидным образом влияет на скорость обучения непосредственным образом.

Пакетная нормализация на каждом слое позволяет существенно ускорить процесс обучения нейронной сети и, что самое важное, позволяет сократить количество эпох в обучении и добиться существенно лучших результатов точности нейронной сети.

РЕГУЛЯРИЗАЦИЯ ОБУЧЕНИЯ НЕЙРОННЫХ СЕТЕЙ

Еще одним подходом не столько для ускорения обучения нейронной сети, сколько для усиления ее обобщающей способности является регуляризация весов при обучении.

Функция нейронной сети очень сложная, сходится она в любом случае в некотором локальном минимуме, но очень важно, чтобы в этом локальном минимуме сохранялась возможность предсказать неизвестные объекты данных.

С обобщающей силой нейросетей в основном и работают все новейшие подходы. Это следует из того, что точность предсказания хорошая, но подавая на вход что-то не очень понятное, нейросеть сразу теряет всю силу предсказания. Ожидается, что нейросеть должна функционировать как мозг.

В регуляризации L_1 задано ограничение на сумму весов:

$$\sum_{i=1}^{N} |w_i| \le \varepsilon.$$

В регуляризации L_2 ситуация очень похожа:

$$\sum_{i=1}^{N} |w_i|^2 \le \varepsilon.$$

Вторая регуляризация работает с квадратами. Первая регуляризация линейная. Вторая больше необходима не для зануления весов, сколько для пенализирования больших выбросов (аномалий).

Регуляризация необходима для выделения множества слабых связей. Она убирает явное переобучение нейронной сети и как бы распределяет распознавание объектов между многими нейронами (то есть, если окажется, что в сложной задаче будет задействован только 1 нейрон из 3, то это все равно лучше, чем 0 нейронов. Такой результат достигается благодаря превентивному выявлению слабых связей).

Также существует термин «Dropout» – переводится как «отсев», но в русскоязычной терминологии особенно не прижился русский аналог. Его методология состоит в том, что пенализируются не только веса, но на каждом шаге будет выбрасываться какое-то количество нейронов из рассмотрения. Благодаря этому достигается случайное обучение нейронов, так как в одну итерацию могут быть исключены из рассмотрения до 50 % всех нейронов в сети. Важные признаки фиксируются в большом количестве нейронов.

Такая технология позволяет бороться с переобучением на статистическом шуме, усиливая обобщающую способность нейронной сети в очень сложных входных условиях (помним, что тренироваться всегда лучше в сложнейших условиях).

Таким образом, прогоняя каждый пакет в итерации, из обучения случайным образом «выбрасываются» определенные нейроны. Это позволяет сильнее «запоминать» признаки, так как сильные признаки чаще встречаются на итерациях. Благодаря этому создаются плотные ассоциативные связи, аналогично структуре человеческого головного мозга.

В большинстве промышленных сборок нейронных сетей Dropout часто используется в финальных стадиях нейронной сети (на этапе принятия решений).

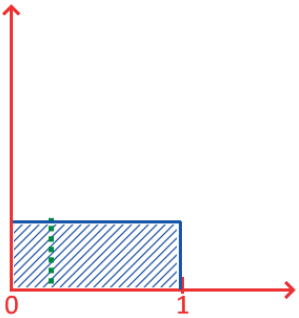
МЕТОДЫ ИНИЦИАЛИЗАЦИИ ВЕСОВ В НЕЙРОННЫХ СЕТЯХ

Еще одним важным параметром, который влияет как на скорость обучения, так и на обобщающую силу нейронной сети, является работа с инициализацией весов.

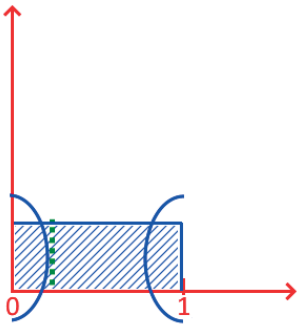
Ранее мы рассмотрели пакетную нормализацию (работает на значения данных; на данные, которые входят на каждый слой). Так вот инициализация весов и функция инициализации весов работают с весами.

Веса инициализируются каким-то случайным образом. Разумеется, что случайный подход допускает равномерно случайный подход. Задать всем нейронам одинаковый вес не имеет никакого смысла, потому что тогда нейросеть становится необучаемой. Для случайной инициализации (начальной инициализации) весов нам нужен некоторый вброс случайных значений.

Первым подходом является равномерная инициализация весов. Она протекает следующим образом: задаем диапазон возможных значений и случайным образом, ***равномерно*** выбираем из него данные.

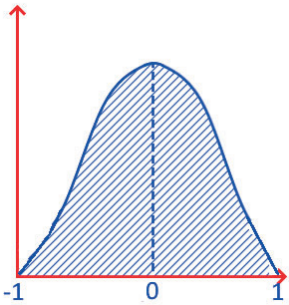


На практике оказалось, что такой несложный подход равномерной инициализации весов работает не очень хорошо. Может оказаться так, что значительная часть весов будет маленькими или значительная часть весов будет большими. То есть, веса могут выбраться в своем большинстве из крайних областей:



Это объяснимо большой зависимостью от начальных значений весов. Отметим, что от начальной инициализации весов зависит очень многое, так как придется многажды запускать нейронную сеть с «нулевого цикла» при плохо подобранных начальных параметрах.

От равномерного подхода перешли к нормальному подходу распределения весов:



По этому распределению легко определить (основываясь на математических аксиомах), что большинство весов будет инициализировано в рамках заданного диапазона (в примере от −1 до 1), попадая в своем большинстве в область значения 0, но при этом имея разные показатели десятичной доли.

Нормальность дает достаточно большую гарантию, что не будет «скученности» весов в каком-то одном месте (по закону нормального распределения «чистое» значение 0 получат только 0,03 % всех нейронов).

В целом это очень хорошая схема предварительной инициализации весов. Однако после 2015 года исследователями из Китая было предложено провести эксперимент по теме среднеквадратичной ошибке среднего, которая неминуемо появляется при инициализации по нормальному распределению. Было определено, что когда весов становится очень много, то схема инициализации весов уже не дает гарантию, так как происходит «раскачка» весов при большом числе нейронов слоя.

Ученые Завьер (также называют Глорот) предложили нормировать веса путем умножения на следующую величину:

$$\delta = \frac{\sqrt{2}}{\sqrt{N_1 + N_2}}.$$

Нормировка полезна тогда, когда на первом слое, например, 10 нейронов, а на втором уже 100. Благодаря этому учитывается разница.

В целом идея вполне понятна: кроме того, что необходима нормировка весов, требуется также и нормирование по числу (нормируем на корень из числа нейронов – умножаем на δ).

Однако, такая нормировка плохо работает для полулинейной функции. Это происходит из-за того, что когда речь идёт про $\sqrt{N}$, то полулинейная функция уже выпадает, потому что она умертвляет половину функции (обращая в ноль, всё, что меньше 0). Исследователь Ге предложил компромисс для полулинейной функции:

$$\mathrm{Re}\, LU : D = \frac{2}{\sqrt{N}}.$$

Если активации следует по полулинейной функции, то веса начальной инициализации весов в слое нейронной сети выполняется по Ге.

Инициализация весов таким образом дает небольшой прирост в точности (1–2 % прироста в точности), но, самым важным является тот факт, что это ускоряет обучение нейронной сети, позволяя добиться лучшей обобщающей силы.

ДОПОЛНЕНИЕ ДАННЫХ

Дополнение (Augmentation) или размножение входных данных является важным методом при работе с нейронной сетью.

Дополнение позволяет существенно улучшить обобщающую силу (в редких случаях – ухудшить результат). Дополнение вытекает из очевидной сложности ручной разметки данных. Многие привычно относятся к тому, что в совершенно любой задаче нужно достичь заданной цели, проверив потом точность модели.

Однако каким образом проверяется точность? Проверкой на тестовых данных. А обучение модели протекает на тренировочных данных. Но каким чудесным образом нейронная сеть узнает, например, что на фото изображена собака, а не голубь? Это достигается при помощи ручной разметки данных людьми, в процессе которого устанавливаются соответствующие параметры того, что в действительности находится в семпле данных. Сами по себе размеченные данные стоят очень дорого, а иногда их не всегда возможно вообще найти. Допустим, рентген редкого вида заболевания можно получить всего от 200 больных по всей планете, в год прибавляется ещё 5 человек. Понятно, что данных недостаточно, а обучать модель необходимо. Именно дополнение данных решает такую часто встречающуюся на практике задачи.

Процесс дополнения может быть осуществлен с любыми данными (фото, аудио, видео, текст и другое). На примере рукописных цифр разберем какие подходы существуют для дополнения данных:

1. Цифру можно повернуть:

2. Цифру можно отразить (зеркальное отражение невозможно получить из поворота, это другой тип аугментации):

Несложно посчитать масштабируемость данных. Даже если поворачивать изображение на 90°, то появляется 4 изображения из одного оригинала (4 вместо 1). Теперь каждый элемент можно ещё и отразить (8 вместо 1).

3. Размытие и коррекция резкости (наложение фильтров резкости) изображений.

4. Внесение дополнительного шума.

На изображении могут присутствовать полосы, дуги и прочее (CAPTCHA).

Данные можно сгенерировать и алгоритмически, но наиболее распространенной технологией является совмещение искусственных и реальных данных. Для этого к оригинальным данным добавляют вариации синтетически созданные, что позволяет получить из 1000 около 15000 данных. Также следует помнить, что по закону увеличения обучающих данных в 15 раз, при ансамблировании алгоритмов, точность нейронной сети может быть повышена в $\sqrt{N} = \sqrt{15}$, что теоретически уменьшает ошибку в 3–5 раз.

Что важнее, повышается обобщающая сила нейронной сети, поскольку каждое входное информационное данное является модифицированной копией оригинала. Это позволяет воссоздать сложные условия и заставляет нейронную сеть обучаться в максимально приближенных к действительным (промышленным) условиям.

СВЕРТКА И ПОДВЫБОРКА

Рассматривая архитектуры нейронных сетей (порядок, комплекция и расположение нейронов, правила их объединения в слои), рассмотрим целый класс нейронных сетей.

Этот класс именуется сверточными нейронными сетями. Класс хорошо изучен и в нем предлагается очень много ценных и практически[3] значимых алгоритмов.

Первый слой сверточной нейронной сети – это сверточный слой. Его изображают некоторой функцией свертки (набор фильтров). Фильтры инициализируются случайным образом и могут быть произвольного размера (точнее сказать произвольного размера может быть матрица фильтра: 3*3, 7*7, 21*21 и так далее).

Пример матрицы фильтра 3*3:

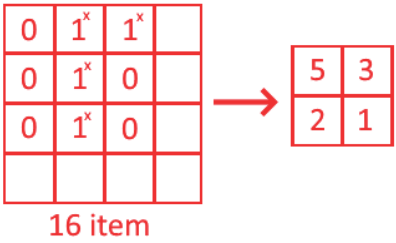
0	1	1	
0	1	0	
0	1	0	

Как позже оказалось, именно такие комбинации единиц и нулей в фильтре, а также сам размер матрицы (3*3) является наиболее удачным в прикладных задачах.

[3] В прикладном понимании.

Далее оперирование происходит матрицами (также их называют ядрами). Ядра фильтруют входные данные и выделяют особенности из данных (например, из изображения).

Веса фильтров умножаются на исходные значения пикселей. Благодаря этой процедуре преобразовывается 16 параметров на первом слое в 4 параметра на втором:



Таким образом, в результате работы ядер над сверточным слоем матрицы N*N параметров, преобразуются в:

$$N * N \rightarrow m * (N-1) * (N-1).$$

Применяя каждую матрицу на квадрате 3*3, мы немножко «схлопнем» пространство, но на самом деле оно размножится: всего на каждом слое применяется 100–200 таких ядер (матриц). Матрицы можно сравнить с пленками диафильмов, поэтому, когда мы каждый раз светим через пленку, некоторые фигуры выделяются и записываются в виде сумм числом.

Но фигур получается много – это одна из проблем свертки. Допустим, из матрицы в 16 каналов, то при m = 8, получается 8*2*2. Каждое из ядер преобразует изображение 4*4 в некоторый квадрат 2*2. Итого на выходе вместо 16 значений параметров получится 32 значений параметров.

Соответственно, одной из проблем, с которой суждено было столкнуться при использовании сверточных нейронных сетей – это мультипликация числа параметров. Однако, ядра существенно помогают в выделение форм через каскад фильтров, что позволяет быстро решить поставленную задачу по сопоставлению форм.

Очевидно, что поиск сложных форм (автомобиль, человек) можно разрешить путем объединения очень многих каскадов в единую форму.

В случае, если нам даже полностью понятен алгоритм формирования изображения из форм, оказывается, что на каждом шаге количество нейронов растет в геометрической прогрессии.

Для избежания данной проблемы был придуман слой подвыборки. У термина существует много названий, в англоязычной литературе установлен термин Pooling Layer. В русскоязычной чаще всего используется термин «Подвыборка».

Подвыборка – это выбор одного элемента из множества. Принято выбирать в большинстве случаев либо MaxPooling, либо AvgPooling (сокращение от Average – среднее). Согласно методологии, рассматривается квадрат 4*4 и из него выбирается максимальный (или средний) элемент и устанавливается, что именно он будет выбран для следующего слоя, а остальные не рассматриваются.

Когда в изображении очень много пикселей, понятно, что каждый конкретный пиксель несет мало информации. Информации больше в группе пикселей, поэтому можно прибегнуть к объединению пикселей (выделять информацию из области пикселей). В большинстве случаев, это существенно снижает алгоритмическую сложность и позволяет быстрее осуществить процесс обучения нейронной сети.

СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ

В прошлой теме было определено, что для вычленения важного результат применяют слой подвыборки (обычно сразу после сверточного слоя или после серии сверточных слоев).

В итоге получается m ядер. Благодаря такой процедуре значение делится пополам:

$$N * N \rightarrow \frac{m * (N-1) * (N-1)}{2}.$$

Основная идея состоит в том, чтобы не сильно наращивать количество нейронов на каждом слое. Например, при изображении в 256*256 пикселей получается примерно 65 тысяч нейронов и нам очень хотелось бы не сильно увеличивать количество нейронов на каждом слое далее.

Поэтому применяем сначала ядра (то есть количество весов в матрице 3*3*128) и потом эту тысячу весов используем дальше на нейронах. Комбинация слоя свертки и подвыборки позволяет достаточно компактно с помощью нейронной сети распознать изображение.

Вся особенность свертки состоит в том, что, мы можем использовать существенно меньше параметров на каждом слое (между слоями, веса). Свертка позволяет не использовать полносвязный слой нейронов (который бы не поместился бы в принципе физически в оперативной памяти) благодаря выделению форм из изображения фильтрами.

Таким образом, за счет существенного уменьшения количества весов и количества параметров, которые нам нужны, чтобы преобразовать последовательно изображения, мы добиваемся того, что у нас сверточная нейронная сеть содержит вменяемое число параметров – она обучаема. И, что самое важное, она умеет классифицировать изображение.

После каскадов (нескольких слоев свертки + подвыборки) полносвязный слой трансформируется по количеству классов. То есть, из выделенных осо-

бенностей всего изображения (4*4=16) – 16 ключевых особенностей, которые являются комбинацией этих геометрических форм, пытаемся сконструировать один из классов, который нам нужно предсказать. Единственное, что нужно понимать: если финальный каскад заканчивается изображением 4*4, то нам нужно будет 16000 весов, чтобы каждый нейрон финального каскада (свертка + подвыборка 4*4) объединить с каждым из классов. Но это тоже относительно не очень много, особенно, если учитывать, что взамен мы получаем небольшое число параметров при обучении.

Отметим, что такая архитектура была изобретена более 40 лет назад, поэтому в практическом понимании она может решать только простые задачи классификации (классификация цифр, например).

ЧАСТЬ ПРАКТИЧЕСКИХ НАВЫКОВ К 8

СВЕРТКА И ПРЕДВЫБОРКА

Постановка задачи

Загрузим подготовленные данные из HDF5.

Разделим данные на обучающие и проверочные в соотношении 80/20.

Используем Keras для построения нейросети с линейным, сверточными слоями и слоями подвыборки. Проверим, какая конфигурация работает лучше линейных слоев.

Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from sklearn.model_selection import train_test_split
from skimage import io
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
os.environ["PATH"] += (os.pathsep +
                       'C:/Program Files (x86)/Graphviz2.38/bin/')
```

```
Using TensorFlow backend.
```

Используемые функции

Ввод [2]:
```
labels = ["Fish", "Flower", "Gravel", "Sugar"]
image_x = 525
image_y = 350
image_ch = 3
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")
```

```
def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice += 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice
```

Загрузка данных

Ввод [3]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [4]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

```
                                          EncodedPixels         Image Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg  Fish
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg  Fish
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg  Fish
12                                                NaN  0035239.jpg  Fish
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg  Fish
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [5]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
                                          EncodedPixels         Image Label
5164   910620 88 910709 3 910713 2 910721 4 910729 16...  3b88744.jpg  Fish
22008                                               NaN  fdfa5e1.jpg  Fish
7568                                                NaN  56609dc.jpg  Fish
19348  66063 663 67463 663 68863 663 70263 663 71663 ...  dfd58e9.jpg  Fish
13120  13653 267 15053 267 16453 267 17853 267 19253 ...  9545812.jpg  Fish
```

Сверточная нейросеть

Подготовим данные для обучения нейросети: будем загружать данные пакетами. Создадим функции для загрузки двух типов данных: графических из изображения и типа облака на изображении.

Ввод [12]:
```
def load_y (df):
    return np.array(df["EncodedPixels"].notnull().astype("int8")).reshape(-1,1)

def load_x (df):
    x = [[]]*len(df)
    for j, file in enumerate(df["Image"]):
        x[j] = io.imread(os.path.join(filesDir, file))
    return np.array(x).reshape(len(df), image_y, image_x, image_ch)
```

```
def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            yield (load_x(df[batch_start:limit]),
                   load_y(df[batch_start:limit]))
            batch_start += batch_size
            batch_end += batch_size
```

Сверточный слой (Conv2D) применяет ядро преобразования (набор математических операций) к области исходного изображения (набора входов с предыдущего слоя) для выделения особенностей (например, определенных фигур - линий или уголков). Принимает в качестве входной формы только двумерные изображения и число цветовых каналов (трехмерный массив данных на 1 изображение).
Сверточный слой "размноживает" исходное изображение: используется заданное (большое) число ядер свертки, которые оптимизируются на этапе обучения нейросети. Поэтому к полученным при свертке данным обычно применяют слой подвыборки (MaxPooling): выделяют самый значимый из квадрата 2x2 или 3x3 элемент, обнаруженный на сверточной слое, чтобы снизить число выходов и ускорить обучение нейросети.

Свертку осуществляем с шагом (strides) 2.

На выходе слоя подвыборки находится двумерный массив нейронов, полученный выборкой из множества преобразований исходного изображения, поэтому его нужно переформировать, перевести в одномерный. Для этого используется плоский слой (Flatten).

Ввод [7]:
```
model = Sequential([
    Conv2D(32, (3,3), input_shape=(image_y, image_x, image_ch),
           strides=(2,2)),
    Activation("relu"),
    Conv2D(32, (3,3), strides=(2,2)),
    Activation("relu"),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Activation("softmax"),
    Dense(1)
])
```

Топология модели

Потребуется graphviz

Ввод [8]:
```
keras.utils.plot_model(model, to_file="ml0041.png",
                       show_shapes=True, show_layer_names=True, rankdir="TB")
```

Ввод [9]:
```
filesDir = "train_images_small"
batch_size = 20
```

Построим и обучим модель, используя адаптивный градиентный спуск и абсолютную ошибку.

Ввод [10]:
```
model.compile(optimizer="adam", loss="mean_absolute_error")
```

Ввод [13]:
```
model.fit_generator(load_data(train, batch_size),
    epochs=100, steps_per_epoch=len(train)//batch_size, verbose=True)
```

```
Epoch 1/100
221/221 [==============================] - 302s 1s/step - loss: 0.4999
Epoch 2/100
221/221 [==============================] - 329s 1s/step - loss: 0.4935
Epoch 3/100
221/221 [==============================] - 284s 1s/step - loss: 0.4858
Epoch 4/100
221/221 [==============================] - 281s 1s/step - loss: 0.4789
Epoch 5/100
221/221 [==============================] - 284s 1s/step - loss: 0.4699
Epoch 6/100
221/221 [==============================] - 316s 1s/step - loss: 0.4618
Epoch 7/100
221/221 [==============================] - 298s 1s/step - loss: 0.4533
Epoch 8/100
221/221 [==============================] - 305s 1s/step - loss: 0.4446
Epoch 9/100
221/221 [==============================] - 283s 1s/step - loss: 0.4371
Epoch 10/100
```

Предсказание значений

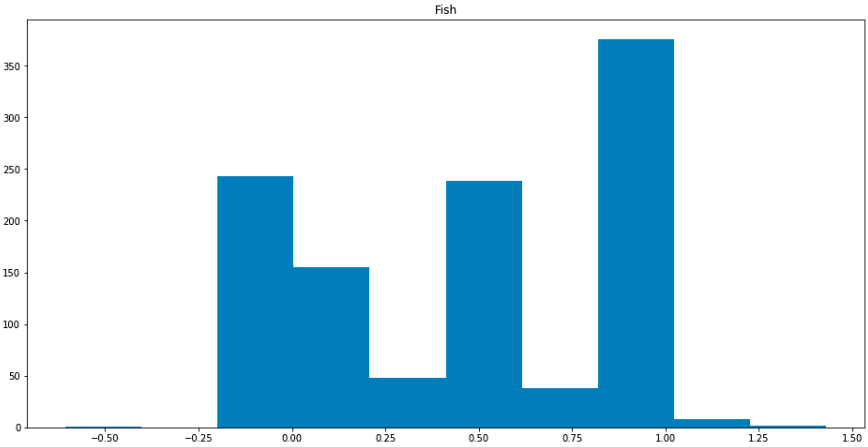
Ввод [14]:
```
prediction = model.predict_generator(load_data(test, 1),
                                     steps=len(test), verbose=1)
```

```
1110/1110 [==============================] - 45s 40ms/step
```

Ввод [15]:
```
def draw_prediction (prediction):
    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1,1,1)
    ax.hist(prediction[0])
    ax.set_title("Fish")
    plt.show()
```

Ввод [16]:
```
prediction = np.transpose(prediction)
```

Ввод [17]:
```
draw_prediction(prediction)
```



Ввод [27]:
```
test["target"] = (prediction[0] >=1).astype("int8")
print (test[test["target"]>0][["EncodedPixels", "target"]].head(20))
```

```
                                            EncodedPixels  target
15144  136115 1019 137515 1019 138915 1019 140315 101...       1
21308                                                NaN       1
15540  538566 179 539966 179 541366 179 542766 179 54...       1
9132                                                 NaN       1
19476  1853664 562 1854227 66 1855064 629 1856464 629...       1
14744                                                NaN       1
1036   429846 842 431246 842 432646 842 434046 842 43...       1
19280                                                NaN       1
6300                                                 NaN       1
16296  300347 653 301747 653 303147 653 304547 653 30...       1
17492                                                NaN       1
9168   1410 223 2810 223 4210 223 5610 223 7010 223 8...       1
21892                                                NaN       1
10304                                                NaN       1
12360  1603080 585 1604480 585 1605880 585 1607280 58...       1
9440   95214 1284 96614 1284 98014 1284 99414 1284 10...       1
9360   2374403 967 2375803 967 2377203 967 2378603 96...       1
7528   115473 583 116873 583 118273 583 119673 583 12...       1
3320                                                 NaN       1
13300                                                NaN       1
```

Оценка по Дайсу

Пока будем считать, что при определении типа облака на изображении, оно целиком размещено на фотографии: т.е. область облака - это все изображение.

Нет облаков - 0.5, MLP - 0.5

Ввод [28]:
```
dice = test.apply(calc_dice, axis=1, result_type="expand")
print ("Kers, (CONV3-32x2,POOL2):", round(dice.mean(), 3))
```

```
Kers, (CONV3-32x2,POOL2): 0.437
```

АКТИВАЦИЯ И ОПТИМИЗАТОРЫ

Постановка задачи

Загрузим данные и разделим их на обучающие и проверочные в соотношении 80/20.

Используем Keras для построения нейросети с линейным, сверточными слоями и слоями подвыборки. Проверим, на что влияют различные функции активации сверточных слоев и какой оптимизатор обеспечивает наилучшую сходимость.

Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from sklearn.model_selection import train_test_split
from skimage import io
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import optimizers
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

```
Using TensorFlow backend.
```

Используемые функции

Ввод [2]:
```
image_x = 350
image_y = 525
image_ch = 3
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()
```

```
def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice += 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice

def load_y (df):
    return np.array(df["EncodedPixels"].notnull().astype("int8")).reshape(len(df), 1)

def load_x (df):
    x = [[]]*len(df)
    for j, file in enumerate(df["Image"]):
        x[j] = io.imread(os.path.join(filesDir, file))
    return np.array(x).reshape(len(df), image_x, image_y, image_ch)

def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            yield (load_x(df[batch_start:limit]),
                   load_y(df[batch_start:limit]))
            batch_start += batch_size
            batch_end += batch_size
```

Загрузка данных

Ввод [23]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [24]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

```
                                          EncodedPixels         Image Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg  Fish
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg  Fish
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg  Fish
12                                                NaN  0035239.jpg  Fish
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg  Fish
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [25]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
                                          EncodedPixels         Image Label
12464                                               NaN  8dc43a0.jpg  Fish
20736  10 1390 1410 1390 2810 1390 4210 1390 5610 139...  eeba036.jpg  Fish
21120                                               NaN  f34fb69.jpg  Fish
14320  152330 3 152336 1 152344 192 153730 3 153736 2...  a4185ec.jpg  Fish
18736                                               NaN  d920035.jpg  Fish
```

Функции активации

Логистическая функция (sigmoid) и гиперболический тангенс (tanh) похожи друг на друга и обеспечивают некоторую плавную сходимость обучения нейросети к нужному результату.

ReLU позволяет провести вычисления быстрее и использовать меньше итераций для обучения, но в случае неудачного выбора начальных коэффициентов возможно "выпадение" отдельных нейронов из слоев и ухудшение предсказательной силы всей нейросети.

Последовательно проверим, как работает обучение при различных функциях активации для первых 50 изображений.

Ввод [16]:
```
def create_model():
    model = Sequential([
        Conv2D(32, (3,3), input_shape=(image_x, image_y, image_ch),
               strides=(2,2)),
        Activation("relu"),
        Conv2D(32, (3,3), strides=(2,2)),
        Activation("relu"),
        MaxPooling2D(pool_size=(2,2)),
        Flatten(),
        Activation("softmax"),
        Dense(1)
    ])
    return model
```

Ввод [11]:
```
filesDir = "train_images_small"
batch_size = 10
```

Построим и обучим модель, используя адаптивный градиентный спуск и абсолютную ошибку.

Ввод [17]:
```
model = create_model()
model.compile(optimizer="adam", loss="mean_absolute_error")
```

Ввод [18]:
```
model.fit_generator(load_data(train[0:50], batch_size),
            epochs=10, steps_per_epoch=len(train[0:50])//batch_size,
            verbose=True)
```

```
Epoch 1/10
5/5 [==============================] - 5s 977ms/step - loss: 0.4216
Epoch 2/10
5/5 [==============================] - 4s 766ms/step - loss: 0.4194
Epoch 3/10
5/5 [==============================] - 4s 801ms/step - loss: 0.4172
Epoch 4/10
5/5 [==============================] - 4s 880ms/step - loss: 0.4157
Epoch 5/10
5/5 [==============================] - 4s 858ms/step - loss: 0.4138
Epoch 6/10
5/5 [==============================] - 4s 741ms/step - loss: 0.4121
Epoch 7/10
5/5 [==============================] - 4s 739ms/step - loss: 0.4106
Epoch 8/10
5/5 [==============================] - 4s 723ms/step - loss: 0.4088
Epoch 9/10
5/5 [==============================] - 4s 746ms/step - loss: 0.4076
Epoch 10/10
5/5 [==============================] - 4s 755ms/step - loss: 0.4058
```

Out[18]: <keras.callbacks.History at 0x23e1d308>

Оптимизаторы нейросети

Проверим работу sgd, nadam, rmsprop. Выставим скорость обучения в 0,02

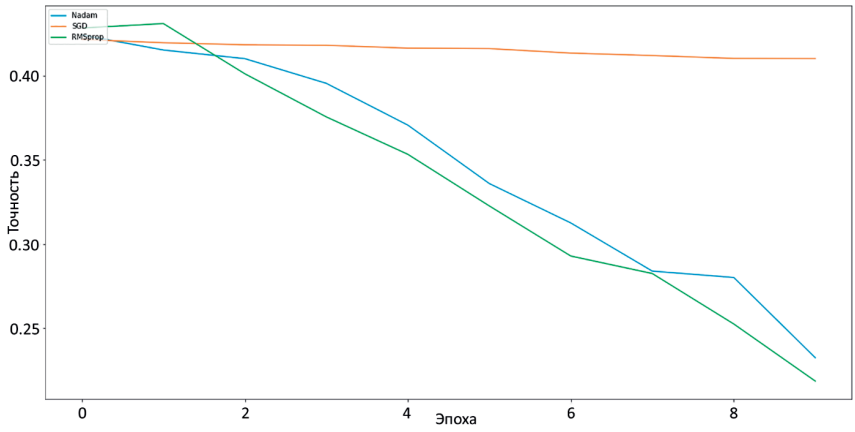
Ввод [20]:
```
history = []
for optimizer in [optimizers.Nadam, optimizers.SGD, optimizers.RMSprop]:
    print (optimizer)
    model = create_model()
    model.compile(optimizer=optimizer(lr=0.02),
                  loss="mean_absolute_error")
    history.append(model.fit_generator(load_data(train[0:50], batch_size),
            epochs=10, steps_per_epoch=len(train[0:50])//batch_size,
            verbose=True).history["loss"])
```

```
<class 'keras.optimizers.Nadam'>
Epoch 1/10
5/5 [==============================] - 5s 930ms/step - loss: 0.4242
Epoch 2/10
5/5 [==============================] - 3s 695ms/step - loss: 0.4153
Epoch 3/10
5/5 [==============================] - 4s 733ms/step - loss: 0.4102
Epoch 4/10
5/5 [==============================] - 4s 702ms/step - loss: 0.3955
Epoch 5/10
5/5 [==============================] - 3s 698ms/step - loss: 0.3707
Epoch 6/10
5/5 [==============================] - 4s 711ms/step - loss: 0.3360
Epoch 7/10
5/5 [==============================] - 4s 716ms/step - loss: 0.3126
Epoch 8/10
5/5 [==============================] - 4s 722ms/step - loss: 0.2839
Epoch 9/10
5/5 [==============================] - 4s 708ms/step - loss: 0.2801
Epoch 10/10
5/5 [==============================] - 4s 712ms/step - loss: 0.2324
<class 'keras.optimizers.SGD'>
Epoch 1/10
5/5 [==============================] - 4s 882ms/step - loss: 0.4216
Epoch 2/10
5/5 [==============================] - 4s 721ms/step - loss: 0.4196
Epoch 3/10
5/5 [==============================] - 4s 724ms/step - loss: 0.4184
Epoch 4/10
5/5 [==============================] - 4s 716ms/step - loss: 0.4181
Epoch 5/10
5/5 [==============================] - 4s 729ms/step - loss: 0.4164
Epoch 6/10
5/5 [==============================] - 4s 707ms/step - loss: 0.4162
Epoch 7/10
5/5 [==============================] - 4s 752ms/step - loss: 0.4135
Epoch 8/10
5/5 [==============================] - 4s 722ms/step - loss: 0.4120
```

```
Epoch 9/10
5/5 [==============================] - 4s 709ms/step - loss: 0.4103
Epoch 10/10
5/5 [==============================] - 4s 703ms/step - loss: 0.4102
<class 'keras.optimizers.RMSprop'>

Epoch 1/10
5/5 [==============================] - 4s 899ms/step - loss: 0.4284
Epoch 2/10
5/5 [==============================] - 4s 730ms/step - loss: 0.4310
Epoch 3/10
5/5 [==============================] - 4s 718ms/step - loss: 0.4012
Epoch 4/10
5/5 [==============================] - 4s 721ms/step - loss: 0.3756
Epoch 5/10
5/5 [==============================] - 4s 713ms/step - loss: 0.3533
Epoch 6/10
5/5 [==============================] - 4s 714ms/step - loss: 0.3227
Epoch 7/10
5/5 [==============================] - 4s 738ms/step - loss: 0.2929
Epoch 8/10
5/5 [==============================] - 4s 706ms/step - loss: 0.2825
Epoch 9/10
5/5 [==============================] - 4s 710ms/step - loss: 0.2525
Epoch 10/10
5/5 [==============================] - 4s 726ms/step - loss: 0.2186
```

Ввод [21]:

```
plt.figure(figsize=(20,10))
for i in range(len(history)):
    plt.plot(history[i])
plt.ylabel("Точность")
plt.xlabel("Эпоха")
plt.legend(["Nadam", "SGD", "RMSprop"], loc="upper left")
plt.show()
```



НОРМАЛИЗАЦИЯ И ПЕРЕОБУЧЕНИЕ

Постановка задачи

Загрузим данные и разделим их на обучающие и проверочные в соотношении 80/20.

Используем Keras для построения нейросети с линейным, сверточными слоями и слоями подвыборки. Проверим, как повышают эффективность обучения нормализация данных и отсев.

Обучим модель, используя последовательную загрузку данных. Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from skimage import io
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten, Dropout
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras import optimizers
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

```
Using TensorFlow backend.
```

Используемые функции

Ввод [2]:
```
image_x = 525 # 525
image_y = 350 # 350
image_ch = 3 # 3
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
```

```
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice += 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice

def load_y (df):
    return np.array(df["EncodedPixels"].notnull().astype("int8")).reshape(len(df), 1)

def load_x (df):
    x = [[]]*len(df)
    for j, file in enumerate(df["Image"]):
        x[j] = io.imread(os.path.join(filesDir, file))
    return np.array(x).reshape(len(df), image_y, image_x, image_ch)
```

```
def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            yield (load_x(df[batch_start:limit]),
                   load_y(df[batch_start:limit]))
            batch_start += batch_size
            batch_end += batch_size

def draw_prediction (prediction):
    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1,1,1)
    ax.hist(prediction[0])
    ax.set_title("Fish")
    plt.show()
```

Загрузка данных

Ввод [3]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [4]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

```
                                          EncodedPixels        Image Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg  Fish
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg  Fish
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg  Fish
12                                                NaN  0035239.jpg  Fish
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg  Fish
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [5]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
                                                EncodedPixels        Image Label
6104                                                      NaN  4669e71.jpg  Fish
8476                                                      NaN  60aa201.jpg  Fish
6216   332608 448 334008 448 335408 448 336808 448 33...  476f864.jpg  Fish
20804  125255 656 126655 656 128055 656 129455 656 13...  eff8789.jpg  Fish
4700                                                      NaN  36198e2.jpg  Fish
```

Генератор данных для обучения

Данные всех изображений могут не поместиться в оперативную память, поэтому будем обучать нейросеть последовательно, пакетами.

Ввод [6]:
```
filesDir = "train_images_small"
batch_size = 50
```

Сверточная нейросеть

Создадим модель. Архитектура в общем виде:

INPUT -> [[CONV -> RELU] * N -> POOL?] * M -> [FC -> RELU] * K -> FC * L

Ввод [7]:
```
model = Sequential([
    Conv2D(32, (3,3), input_shape=(image_y, image_x, image_ch),
           kernel_initializer='glorot_uniform', strides=(2,2)),
    Activation("relu"),
    BatchNormalization(),
    Dropout(0.5),
    Conv2D(32, (3,3),
           kernel_initializer='glorot_uniform', strides=(2,2)),
    Activation("relu"),
    BatchNormalization(),
    Dropout(0.5),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Activation("softmax"),
    Dense(1)
])
```

```
WARNING: Logging before flag parsing goes to stderr.
W0314 21:49:40.229383  9288 deprecation_wrapper.py:119] From c:\users\nikolay\appdata\local\programs\python\python37\
lib\site-packages\keras\backend\tensorflow_backend.py:1834: The name tf.nn.fused_batch_norm is deprecated. Please use
tf.compat.v1.nn.fused_batch_norm instead.

W0314 21:49:40.349390  9288 deprecation.py:506] From c:\users\nikolay\appdata\local\programs\python\python37\lib\site
-packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_pro
b is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

Ввод [8]:
```
model.compile(optimizer=optimizers.Nadam(lr=0.02),
              loss="mean_absolute_error")
```

Предсказание значений

Ввод [9]:
```
model.fit_generator(load_data(train, batch_size),
                    epochs=50, steps_per_epoch=len(train)//batch_size)
```

```
Epoch 1/50
88/88 [==============================] - 1357s 15s/step - loss: 0.5035
Epoch 2/50
88/88 [==============================] - 1356s 15s/step - loss: 0.5060
Epoch 3/50
88/88 [==============================] - 1359s 15s/step - loss: 0.5054
Epoch 4/50
88/88 [==============================] - 1267s 14s/step - loss: 0.5129
Epoch 5/50
88/88 [==============================] - 1211s 14s/step - loss: 0.5120
Epoch 6/50
88/88 [==============================] - 1204s 14s/step - loss: 0.5142
Epoch 7/50
88/88 [==============================] - 1207s 14s/step - loss: 0.5341
Epoch 8/50
88/88 [==============================] - 1216s 14s/step - loss: 0.5228
```

```
Epoch 9/50
88/88 [==============================] - 1206s 14s/step - loss: 0.5180
Epoch 10/50
88/88 [==============================] - 1213s 14s/step - loss: 0.5178
Epoch 11/50
88/88 [==============================] - 1208s 14s/step - loss: 0.5075
Epoch 12/50
88/88 [==============================] - 1206s 14s/step - loss: 0.5372
Epoch 13/50
88/88 [==============================] - 1200s 14s/step - loss: 0.5348
Epoch 14/50
88/88 [==============================] - 1198s 14s/step - loss: 0.5355
Epoch 15/50
88/88 [==============================] - 1199s 14s/step - loss: 0.5233
Epoch 16/50
88/88 [==============================] - 1231s 14s/step - loss: 0.5181
Epoch 17/50
88/88 [==============================] - 1199s 14s/step - loss: 0.5197
Epoch 18/50
88/88 [==============================] - 1202s 14s/step - loss: 0.5229
Epoch 19/50
88/88 [==============================] - 1228s 14s/step - loss: 0.5336
Epoch 20/50
88/88 [==============================] - 1198s 14s/step - loss: 0.5232
Epoch 21/50
88/88 [==============================] - 1201s 14s/step - loss: 0.5082
Epoch 22/50
88/88 [==============================] - 1203s 14s/step - loss: 0.5109
Epoch 23/50
88/88 [==============================] - 1196s 14s/step - loss: 0.5047
Epoch 24/50
88/88 [==============================] - 1202s 14s/step - loss: 0.5033
Epoch 25/50
88/88 [==============================] - 1205s 14s/step - loss: 0.5044
Epoch 26/50
88/88 [==============================] - 1206s 14s/step - loss: 0.5051
Epoch 27/50
88/88 [==============================] - 1204s 14s/step - loss: 0.5054
Epoch 28/50
88/88 [==============================] - 1204s 14s/step - loss: 0.4989
Epoch 29/50
88/88 [==============================] - 1204s 14s/step - loss: 0.5023
Epoch 30/50
88/88 [==============================] - 1202s 14s/step - loss: 0.5040
Epoch 31/50
88/88 [==============================] - 1203s 14s/step - loss: 0.5070
Epoch 32/50
88/88 [==============================] - 1201s 14s/step - loss: 0.4926
Epoch 33/50
88/88 [==============================] - 1198s 14s/step - loss: 0.4967
Epoch 34/50
88/88 [==============================] - 1244s 14s/step - loss: 0.4966
Epoch 35/50
88/88 [==============================] - 1239s 14s/step - loss: 0.5002
Epoch 36/50
88/88 [==============================] - 1223s 14s/step - loss: 0.4877
Epoch 37/50
88/88 [==============================] - 10088s 115s/step - loss: 0.5281
Epoch 38/50
88/88 [==============================] - 1423s 16s/step - loss: 0.5525
Epoch 39/50
88/88 [==============================] - 5749s 65s/step - loss: 0.5086
Epoch 40/50
88/88 [==============================] - 1317s 15s/step - loss: 0.4960
```

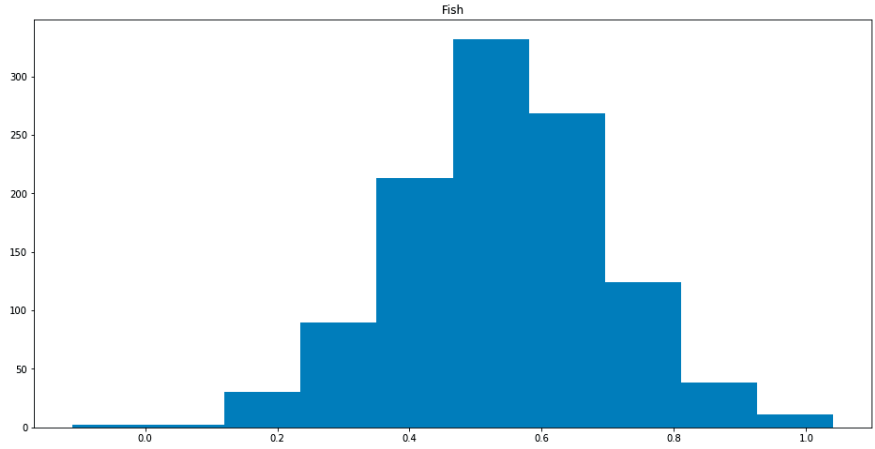
```
Epoch 41/50
88/88 [==============================] - 1366s 16s/step - loss: 0.4913
Epoch 42/50
88/88 [==============================] - 1276s 15s/step - loss: 0.4944
Epoch 43/50
88/88 [==============================] - 1330s 15s/step - loss: 0.4992
Epoch 44/50
88/88 [==============================] - 1230s 14s/step - loss: 0.4905
Epoch 45/50
88/88 [==============================] - 1285s 15s/step - loss: 0.4931
Epoch 46/50
88/88 [==============================] - 1842s 21s/step - loss: 0.5003
Epoch 47/50
88/88 [==============================] - 4448s 51s/step - loss: 0.4879
Epoch 48/50
88/88 [==============================] - 11950s 136s/step - loss: 0.4976
Epoch 49/50
88/88 [==============================] - 1741s 20s/step - loss: 0.4937
Epoch 50/50
88/88 [==============================] - 1473s 17s/step - loss: 0.4868

<keras.callbacks.History at 0x23df7e48>
```

Ввод [10]:
```
prediction = model.predict_generator(load_data(test, 1),
                                     steps=len(test), verbose=1)
```

```
1110/1110 [==============================] - 148s 133ms/step
```

Ввод [11]:
```
prediction = np.transpose(prediction)
```

Ввод [12]:
```
draw_prediction(prediction)
```



Ввод [20]:
```
test["target"] = (prediction[0] >= 0.95).astype("int8")
print (test[test["target"]>0][["EncodedPixels","target"]].head(100))
```

```
                                            EncodedPixels  target
19652  1124605 556 1126005 556 1127405 556 1128805 55...       1
15796                                                NaN       1
16904  851201 395 851597 2 851609 8 852601 400 853009...       1
12196                                                NaN       1
21716  96 777 1496 777 2896 777 4296 777 5696 777 709...       1
14268  845 447 2245 447 3645 447 5045 447 6445 447 78...       1
13092                                                NaN       1
18664                                                NaN       1
```

Оценка по Дайсу

Пока будем считать, что при определении типа облака на изображении, оно целиком размещено на фотографии: т.е. область облака - это все изображение.

Нет облаков - 0.5, MLP - 0.3, CONV3-32x2,POOL2 - 0.46

Ввод [21]:
```
dice = test.apply(calc_dice, axis=1, result_type="expand")
print ("Keras, (CONV3-32x2,POOL2):", round(dice.mean(), 3))
```

```
Keras, (CONV3-32x2,POOL2): 0.488
```

ДОПОЛНЕНИЕ ИЗОБРАЖЕНИЙ

Постановка задачи

Загрузим уменьшенные изображения, уменьшим их еще в 2.5 раза и применим к ним дополнение (augmentation): смещение, контрастность, поворот, размытие.

Загрузим данные и разделим их на обучающие и проверочные в соотношении 80/20.

Используем Keras для построения нейросети с линейным, сверточными слоями и слоями подвыборки.

Обучим модель, используя последовательную загрузку данных. Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from scipy import ndimage
from skimage import transform,util,exposure,io,color
from sklearn.model_selection import train_test_split
from keras.models import Sequential
```

```
from keras.layers import Dense, Activation, Flatten, Dropout
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras import optimizers
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

```
Using TensorFlow backend.
```

Используемые функции

Ввод [28]:

```
image_x = 210 # 525
image_y = 140 # 350
image_ch = 1 # 3
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice += 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice

def load_y (df):
    return np.array(df["EncodedPixels"].notnull().astype("int8")).reshape(len(df), 1)

def load_x (df):
    x = [[]]*len(df)
    for j, file in enumerate(df["Image"]):
        x[j] = io.imread(os.path.join(filesDir, file))
    return np.array(x).reshape(len(df), image_y, image_x, image_ch)

def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            yield (load_x(df[batch_start:limit]),
                   load_y(df[batch_start:limit]))
            batch_start += batch_size
            batch_end += batch_size

def draw_prediction (prediction):
    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1,1,1)
    ax.hist(prediction[0])
    ax.set_title("Fish")
    plt.show()
```

Обработка и дополнение изображений

Преобразуем изображение: уменьшим в 2,5 раза, добавим случайный шум, повернем на 5 градусов по часовой и против часовой стрелки, увеличим контрастность, изменим гамму и добавим размытие по 3 и 7 пикселям

Ввод [7]:
```
def img_save (name, data):
    io.imsave(os.path.join(filesDir, name),
              (data*255).astype(np.uint8))
def img_aug (name):
    if not os.path.isdir(filesDir):
        os.mkdir(filesDir)
    img = io.imread(os.path.join(dir_, name))
    img = transform.rescale(img, 1 / 2.5)
    img_save(name, img)
    img_save("noised_" + name, util.random_noise(img))
    img_save("rotcw_" + name, transform.rotate(img, -5))
    img_save("rotccw_" + name, transform.rotate(img, 5))
    v_min, v_max = np.percentile(img, (0.2, 99.8))
    img_save("cont_" + name, exposure.rescale_intensity(img,
                                in_range=(v_min, v_max)))
    img_save("gamma_" + name, exposure.adjust_gamma(img,
                                gamma=0.4, gain=0.9))
    img_save("blurred3_" + name,
             ndimage.uniform_filter(img, size=(3,3,1)))
    img_save("blurred7_" + name,
             ndimage.uniform_filter(img, size=(7,7,1)))
```

Обработаем все изображения обучающей выборки

Ввод [5]:
```
dir_ = "train_images_small"
filesDir = dir_ + "_tiny"
```

Ввод [8]:
```
for file in os.listdir(dir_):
    img_aug(file)
```

Загрузка данных

Ввод [9]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [10]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

```
                                          EncodedPixels        Image Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg  Fish
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg  Fish
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg  Fish
12                                                NaN  0035239.jpg  Fish
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg  Fish
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [11]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
                                          EncodedPixels        Image Label
16844  373 500 1773 500 3173 500 4573 500 5973 500 73...  c2ad0d8.jpg  Fish
18724                                                NaN  d90e620.jpg  Fish
6816   423255 412 424655 412 426055 412 427455 412 42...  4e60552.jpg  Fish
1064                                                 NaN  0ca6ecd.jpg  Fish
9928                                                 NaN  7100409.jpg  Fish
```

Дополнение обучающих данных

Дополним все измененные изображения типами и областями облаков

Ввод [12]:
```
train.set_index("Image", inplace=True)
fileList = os.listdir(filesDir)
for file in fileList:
    img = file.split("_")
    if (file not in train.index.values and len(img) > 1 and
       img[1] in train.index.values):
        train.loc[file] = [train.loc[img[1]]["EncodedPixels"], "Fish"]
train.reset_index(inplace=True)
print (train.head())
```

```
          Image                                       EncodedPixels Label
0  c2ad0d8.jpg  373 500 1773 500 3173 500 4573 500 5973 500 73...  Fish
1  d90e620.jpg                                                NaN  Fish
2  4e60552.jpg  423255 412 424655 412 426055 412 427455 412 42...  Fish
3  0ca6ecd.jpg                                                NaN  Fish
4  7100409.jpg                                                NaN  Fish
```

Сверточная нейросеть

Создадим и построим модель

Ввод [13]:
```
model = Sequential([
    Conv2D(32, (3,3), input_shape=(image_y, image_x, image_ch),
           kernel_initializer="glorot_uniform", strides=(2,2)),
    Activation("relu"),
    Conv2D(32, (3,3),
           kernel_initializer="glorot_uniform", strides=(2,2)),
    Activation("relu"),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Activation("softmax"),
    Dense(1)
])
```

Ввод [14]:
```
model.compile(optimizer=optimizers.Nadam(lr=0.02),
              loss="mean_absolute_error")
```

Обучение модели

Используем для обучения все изображения, включая измененные

Ввод [15]:
```
batch_size=100
```

Ввод [16]:
```
model.fit_generator(load_data(train, batch_size),
                    epochs=100, steps_per_epoch=len(train)//batch_size)
```

```
Epoch 1/100
354/354 [==============================] - 483s 1s/step - loss: 0.4640
Epoch 2/100
354/354 [==============================] - 252s 712ms/step - loss: 0.4807
Epoch 3/100
354/354 [==============================] - 250s 706ms/step - loss: 0.4722
Epoch 4/100
354/354 [==============================] - 250s 707ms/step - loss: 0.4618
Epoch 5/100
354/354 [==============================] - 251s 708ms/step - loss: 0.4616
```

```
Epoch 6/100
354/354 [==============================] - 255s 720ms/step - loss: 0.4633
Epoch 7/100
354/354 [==============================] - 247s 697ms/step - loss: 0.4607
Epoch 8/100
354/354 [==============================] - 248s 702ms/step - loss: 0.4565
Epoch 9/100
354/354 [==============================] - 248s 700ms/step - loss: 0.4642
Epoch 10/100
```

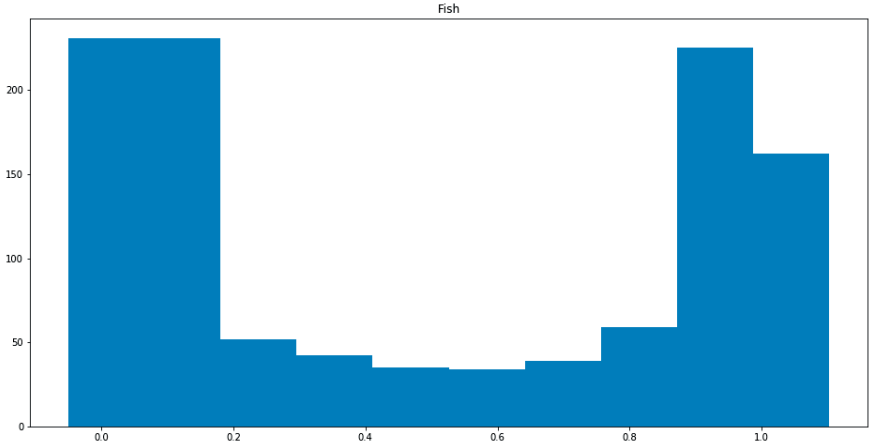
Предсказание значений

Ввод [17]:
```
prediction = model.predict_generator(load_data(test, 1),
                                     steps=len(test), verbose=1)
```

```
1110/1110 [==============================] - 20s 18ms/step
```

Ввод [19]:
```
prediction = np.transpose(prediction)
```

Ввод [20]:
```
draw_prediction(prediction)
```



Ввод [26]:
```
test["target"] = (prediction[0] >= 1).astype("int8")
print(test[test["target"]>0][["EncodedPixels","target"]].head(100))
```

```
                                          EncodedPixels  target
21880  946430 330 947830 330 949230 330 950630 330 95...       1
7960                                                 NaN       1
4292   158666 31 158698 6 160066 19 160086 1 160088 1...       1
21544  602756 314 604153 317 605526 1 605529 1 605532...       1
16888  450101 559 451501 559 452901 559 454301 559 45...       1
11092                                                NaN       1
548    564 535 1101 2 1104 1 1964 535 2501 1 3364 535...       1
11720                                                NaN       1
6184                                                 NaN       1
16660  791722 278 793122 278 794522 278 795922 278 79...       1
8      3510 690 4910 690 6310 690 7710 690 9110 690 1...       1
4808   830320 891 831720 891 833120 891 834520 891 83...       1
13308                                                NaN       1
19936                                                NaN       1
12                                                   NaN       1
6500   321050 271 321327 2 321331 6 322450 275 322727...       1
15924  23809 1367 25209 1367 26609 1367 28009 1367 29...       1
18996                                                NaN       1
```

Оценка по Дайсу

Пока будем считать, что при определении типа облака на изображении, оно целиком размещено на фотографии: т.е. область облака - это все изображение.

Нет облаков - 0.5, MLP - 0.3, CONV3-32x2,POOL2 - 0.48

Ввод [29]:
```
dice = test.apply(calc_dice, axis=1, result_type="expand")
print ("Keras, (CONV3-32x2,POOL2):", round(dice.mean(), 3))
```

```
Keras, (CONV3-32x2,POOL2): 0.451
```

Часть 9
АРХИТЕКТУРЫ СВЕРТОЧНЫХ НЕЙРОСЕТЕЙ

LENET

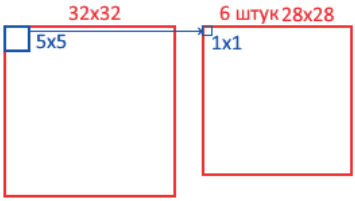
Продолжая исследование сверточных нейросетей, рассмотрим каким образом они эволюционировали от распознавания простых цифр до выделения сложных многоцветных объектов на больших изображениях и локализации местоположения этих объектов на изображениях.

Началось всё с успешной первой сверточной нейросети, разработанной в середине девяностых годов 20 века Яном Лекуном. Она называлась LeNet по сокращению фамилии автора. Эта сеть позволила на ограниченных тогда аппаратных ресурсах построить и успешно обучить нейросеть которая отлично классифицировала чёрно-белые начертания цифр, практически так же точно, как человек.

В то время это был прорыв в использовании нейросетей и послужило отправной точкой для успешного внедрения дальнейших сверточных нейросетей. Разберем принцип работы этой нейросети. На вход сети подавались изображения цифр, всего десять классов и одиннадцатый класс обозначал отсутствие цифры, размер изображения был 32×32 пикселя. Пиксели черно-белые, то есть либо цвет присутствует, либо цвета нет. По этим изображения двигается фильтр 5×5, например:

$$\begin{matrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{matrix}$$

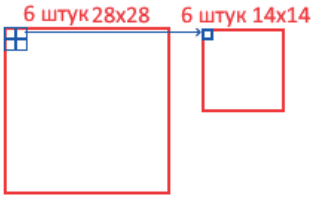
В данном случае фильтр выделяет некоторую вертикальную черту. Понятно, что он может выделять горизонтальную черту, крестик, уголки, квадрат. Лекун предложил использовать эти шесть фильтров 5×5, чтобы из исходного изображения 32×32 получить 6 проекций 28×28:



Каждая область 5×5 накладывается на исходные пиксели. То есть этот фильтр как бы проявляет эту вертикальную полоску на изображении, если черные пиксели наложились на вертикальную полоску, то выходит максимальная сумма и он переходит в левый верхний пиксель. Сдвигая фильтр на единицу, получаем следующие значения.

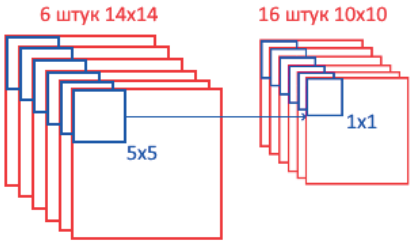
Получается, что матрица свертки — это некий фильтр, представляющий собой набор чисел, который накладываем на области исходного изображения, перемножаем с значениями, например цвета исходного изображения и затем Света, например области исходного изображения затем складываем перемноженные значения в одно число. Это и есть свертка, 25 пикселей сворачиваем в одно значение. Следующие 25 в другое значение и в конце получаем 28×28, но уже шесть проекций. Поскольку мы применяем шесть разных вариантов фильтров. Получается, что из начальных 25 значений весов, мы уже получили $25 * 6 = 150$ значений.

Понятно, что соседние пиксели содержат мало различной информации об исходном изображении. Поэтому после слоя свертки используют слой подвыборки. Слой подвыборки заключается в том, что мы сворачиваем каждые четыре пикселя, то есть выбираем один из них:

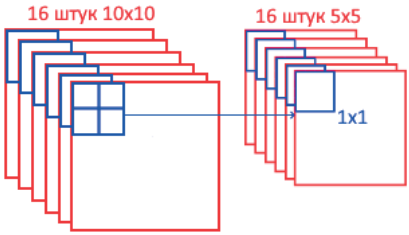


У нас остается шесть проекций, но размер изображения сократился до 14×14. Следующий шаг сводится к применению нового набора фильтров, состоящего уже из шестнадцати вариантов. Эти фильтры складываются из базовых фильтров, например горизонтальная линия с крестиком, либо вертикальная линия с уголком. Получается шестнадцать различных наборов этих фильтров.

Размер матрицы свертки остается таким же 5×5. После наложения этой матрицы на изображение 14×14 получаем изображение 10×10. Важно помнить, когда применяем фильтр, то он применяется ко всем шести проекциям:



Всеми этими процессами мы стремимся получить максимально разнообразные представления исходной комбинации. Далее снова идет слой подвыборки:



Мы свернули четыре пикселя в один и получили шестнадцать изображений 5×5. Шестнадцать изображений 5×5 пикселей дают нам $25 * 16 = 400$ значений, которые представляют собой комбинацию простых форм фильтров и подвыборку из наиболее существенных значений этих простых форм.

В конце присоединяем 400 пикселей к 120 нейронам, затем к 84 нейронам и все это переходит в 10 классов. Получается в архитектуре LeNet имеется два полносвязных слоя в конце. Таким образом выходят два каскада свертки подвыборки, то есть вот последовательно выделяем простые формы, потом комбинации этих простых форм и затем уже через два полносвязных слоя пытаемся, до перемножить найденные особенности и успешно выделяем в них 10 классов.

Как показала практика сеть достаточно успешно смогла сжать изображение 32×32 до 10 классов. LeNet первая успешная сверточная нейросеть, которая позволила успешно решить задачу распознавания рукописных цифр. Это положило начало применению сверточных архитектур. На сегодняшний день сверточные нейросети используются повсеместно: мобильными телефонами, умными часами, камерами с распознаванием лиц и так далее.

ALEXNET

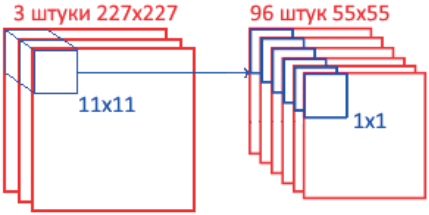
Следующим большим шагом в использовании сверточных нейросетей, стала нейросеть AlexNet, созданная в 2012 году. Она позволила работать с достаточно большими полноцветными изображениями и классифицировать их по 1000 разных классов. Это были некоторые объекты реального мира: машины, самолёты, животные, и так далее всего 1000 классов. Несколько миллионов или даже десятков миллионов изображений, на которых нейросеть обучалась.

Это был большой фундаментальный прорыв, поскольку эта архитектура сверточной нейросети позволила решить большинство промышленных задач.

Например: подавая на вход этой архитектуры базу лиц рабочих предприятия, на выходе получаем возможность внедрения на предприятии распознавания лиц по фотографиям.

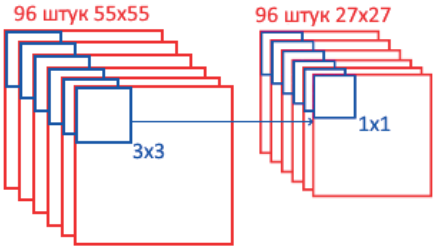
Рассмотрим, как организована нейросеть AlexNet. На вход подается изображение 227×227. Такой размер обусловлен архитектурой. Нужно получить целое число пикселей на каждом шаге. Это не значит, что она не могла бы использовать изображения других размеров, но это требует корректировки архитектуры. Для текущих архитектур сверточных нейросетей такой проблемы нет.

Также это не просто черно-белое изображение, она работает с тремя цветовыми каналами, для примера будем называть их RGB. Накладываем сверточный фильтр 11×11:

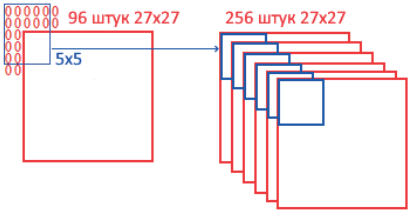


Свертка работает для всех трёх изображений, то есть суммируется не $11*11=121$ значений, а $3*11*11=363$ значений в один новый пиксель. После наложения всех фильтров получится 96 изображений 55×55. 96 изображений получаются из-за сдвига фильтра. В отличии от LeNet, он равен 4.

Далее идет слой подвыборки:

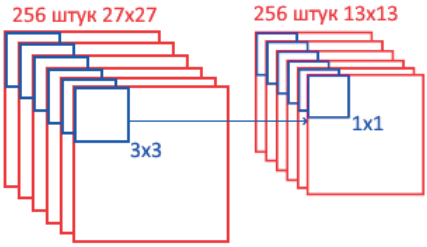


За счёт слоя подвыборки 3×3 сворачиваем к одному пикселю. Сдвиг слоя равен двум пикселям. Дальше применяем сверточный фильтр 5×5, он также применяется одновременно на все 96 изображений. Также важная часть архитектуры AlexNet, применяем фильтр не с левого верхнего угла, а делаем «рамку» из нулей в два пикселя вокруг изображения:



Этим мы пытаемся учесть все пиксели, не уменьшая размер изображения. Получается фильтр 5×5 даст нам также изображение 27×27, только число изображений станет 256. И за счет сдвига на 2 пикселя во все стороны, применяя фильтр 5×5 не уменьшили число исходных пикселей.

Дальше снова идет слой подвыборки 3×3, с сдвигом в два пикселя:



Из изображений 27×27 получаем 13×13. потом мы применяем ещё последовательно три слоя подвыборки с фильтрами:

1) 384 фильтра 3*3 с отступом 2;
2) 384 фильтра 3*3 с отступом 2;
3) 256 фильтров 3*3 с отступом 2.

Таким образом, мы сохраняем размер изображения, но пытаемся перемножить все варианты найденных форм.

После этого, мы получаем многообразие сверток. Далее следует 2 полносвязных слоя по 4096 нейронов. Затем следует 1000 классов (образуя тем самым выходной слой Softmax в 1000 классов).

Важной особенностью AlexNet является использование при обучении полулинейной функции активации (ReLU), вместо арктангенса или сигмоиды (как в случае с LeNet). Применение полулинейной функции позволило существенно ускорить процесс и не потерять при этом точность. Единственное, отметим, что сигмоида всё-таки используется, но только в SoftMax.

Такая конфигурация позволила в 2012 году эффективно решить задачу классификацию в достаточно большом разрешении (200–250 пикселей), при этом уровень ошибки в распознавании несколько объектов составлял точность, сравнимую с точностью человека.

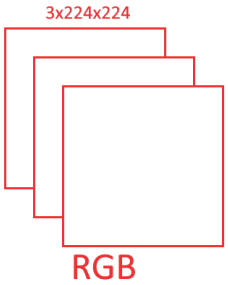
VGG

После появления сверточной архитектуры AlexNet, развитие сверточных нейросетей пошло семимильными шагами. Базовые методы и подходы AlexNet неоднократно развивались и модифицировались для более эффективного решения задач.

Одной из ветвей этого развития это VGG. Она обошла AlexNet в 2014 году на задачах распознавания изображений, но на текущий момент это ветка архитектуры является тупиковой. Разберемся почему.

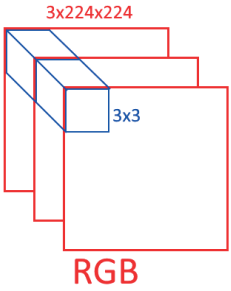
VGG – это некоторое максимальное доведение до предела принципов, заложенных в AlexNet. Сначала берется изображение, на него накладывается ряд фильтров одного размера, потом ряд фильтр другого размера, потом третьего. Тем самым уменьшается изображение, выделяется набор фильтров, а потом из этих наборов фильтров (достаточно высокого уровня) свертываем изображение до 1000 классов.

Посмотрим, как устроена VGG и как она работает с изображением. На вход подается точно такое же (как и в прошлом примере) фото в формате трёхканального изображения 224*224 пикселей.



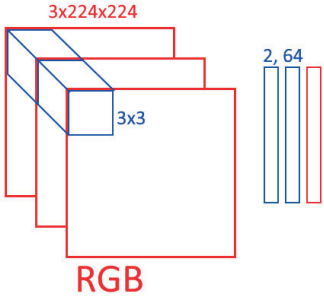
Далее мы применяем три эти изображения на 2 набора сверточных слоев по 64 ядра каждый.

Получается следующий первый слой: 64 фильтра 3*3 с отступом 2, чтобы сохранить размер изображения:



Необходимо 2 каскада этих слоев, потому что было показано, что несколько каскадов сверток работают как одна большая свертка (вместо слоя 5*5 можно использовать два каскада 3*3 и это позволяет уменьшить количество слоев, требуемых для эффективной работы).

Кроме того, нам необходим слой подвыборки:

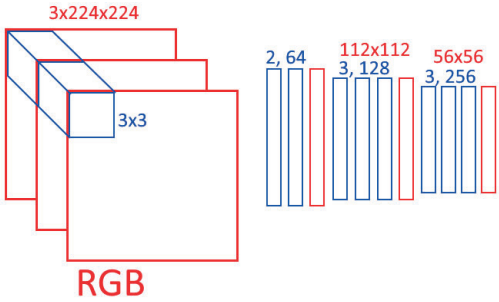


В VGG используется подвыборка 2 на 2, то есть изображение 224*224 переводит в изображение в 112*112.

После этого, на изображение (которое составляет уже 112*112) накладывается последовательно три слоя сверток по 3*3. Обучение фильтров нейросетью происходит под конкретную поставленную задачу (нейронная сеть заранее «не знает» какие формулы и веса лучше подойдут, поэтому обучается по методу обратного распространения ошибки.

Первый каскад содержит две свертки по 64 ядра 3*3, второй каскад содержит уже 3 свертки, но в нем уже присутствует некоторая вариативность, зависящая от архитектуры (поскольку VGG – это семейство нейросетей; в зависимости от силы семейства могут быть 2, 3 или даже 4 сверточных слоя).

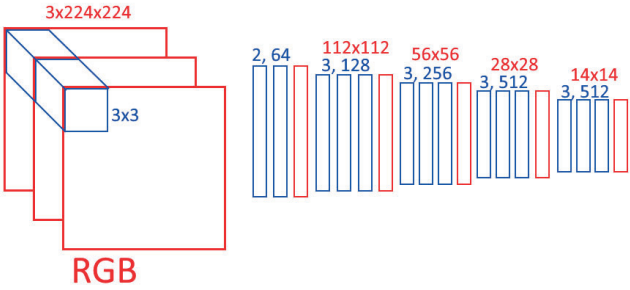
В базовом варианте каскад состоит из 3 сверток по 128 фильтров на каждом слое, применяемым на изображение 112*112. За ним следует слой подвыборки 56*56. 56*56 также применяет ещё две или три свертки 3*3 и «схлопывает» до 28*28.



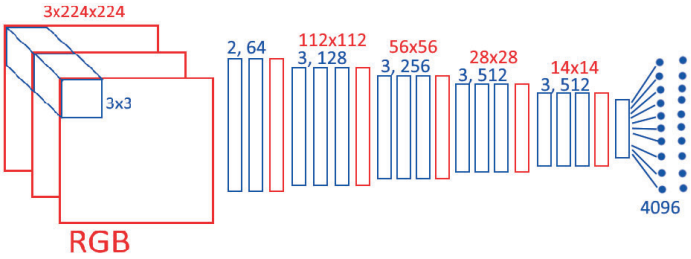
Таким образом, применяя три сверточных слоя 3*3 мы получаем фильтр 9*9, однако используем меньшее число параметров (меньшего числа весов).

Четвертый каскад сверток составляет 2–4 слоя по 512 фильтров. Изображение сворачивается четвертой подвыборкой от 28*28 до 14*14. На 5 это уже 7*7.

Общий схематичный вид представления последовательности каскадов изображен на рисунке ниже:



Слой 7*7 является финальным слоем, и он уже связан с полносвязными слоями. После него следуют полносвязные слои по 4096 нейронов (два таких слоя).



На полносвязный слой выходит 7*7*512 пикселей (по числу нейронов). Далее мы стыкуем их полносвязным образом с первым слоем на 4096 нейронов, потом со вторым слоем на 4096 нейронов и только потом идет финальная стыковка с Softmax на 1000 классов.

Таким образом, имея на входе порядка 10 000–20 000 параметров, можно схлопнуть их до 1000 классов благодаря архитектуре VGG. Каждый класс будет характеризоваться некоторыми 20 параметрами (каждый из этих параметров – это некоторая комбинация форм, комбинация комбинаций форм и так далее). То есть каждый из каскадов делает комбинацию и выборку из наших форм.

Архитектура VGG позволила в 2014–2015 году получить максимальную точность на задаче классификации (распознавания) изображений, но, к сожале-

нию, явилась тупиковой веткой в развитии сверточных нейронных сетей. Почему VGG стало тупиком в развитии, будет обосновано в следующих разделах.

GOOGLENET

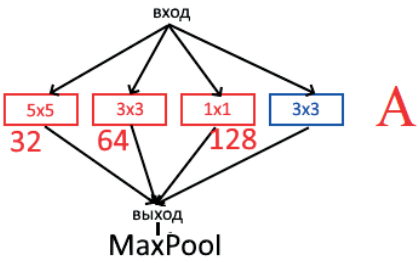
После успеха сети AlexNet в распознавании изображений достаточно большого количества классов, то есть уже промышленная классификация изображений. На базе архитектуры AlexNet было предложено несколько вариантов развития, некоторые из них используются в настоящее время. Один из них – это сеть GoogLeNet или другое название Inception v1. Архитектуры Inception показывают, насколько глубоко мы можем зайти в обучении сверточных нейросетей.

Сейчас уже достигли практически максимально достижимую глубину, потому что при обучении нейросетей обратное распространение ошибок замедляется и становится много более сложным, если у нас много сверточных слоёв. Когда их пять, десять, двадцать слоёв, мы ещё можем нейросеть обучить на простых классах, но когда пытаемся добавить больше, то происходит сильная раскачка весов, поскольку нейросеть на небольшое изменение в классификации должна распространить в самое начало, то есть нужно как-то усилить это небольшое изменения, чтобы оно дошло до начала весов и по пути оно может разбалансировать всё достигнутое обучение.

Именно эта проблема раскачки весов при обучении в случаи глубоких сверточных нейросетей непреодолима. Поэтому сейчас ищут архитектуры, которые позволили бы расширять сеть не в глубину, а в ширину. Глубина нейросети примерно ограничена 100 слоями. Больше чем 100 слоёв на текущих серверных мощностях недостижимо.

Вернемся к GoogLeNet. В чём состоят основные идеи этой архитектуры, почему она оказалась прорывной. Вспомним что такое сверточный блок. Это одна или несколько сверток, расположенных последовательно и позволяющая каким-то образом выделить контурную особенность из изображений и объединить их с помощью слоя подвыборки.

Основная идея сверточный блок Inception v1 заключается в параллельном расположении слоев. Производится параллельная свертка по 5×5, 3×3, 1×1, а также уменьшение изображения слоем MaxPooling подвыборки 3×3:

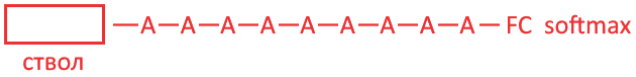


На вход подаем одни и те же данные, они вычисляют какую-то информацию, и она объединяется. MaxPooling уменьшает изображение в 4 раза. Выходы, которые мы получаем от этих фильтров объединяем, то есть складываем, а не перемножаем. Вся эта ячейка работает как некоторый слой свертки.

Это основная идея GoogLeNet и серии сети Inception. То есть мы наращиваем сложность наращиваем сверточные слои не вглубь, как в случае VGG, а вширь. Эта ячейка A легла в основу GoogLeNet и в ней таких ячеек 9. Это не единственная идея, которая лежит в основе современных сверточных нейросетей.

Вторая не менее важная идея. Мы можем как-то предобработать изображение. В сети AlexNet, LeNet, VGG у нас был некоторый быстро сужающий размер изображения блок. В LeNet был 5*5, в AlexNet 11*11 фильтр, таким образом большое изображение очень быстро схлопываем до небольшого и потом уже с ним работаем.

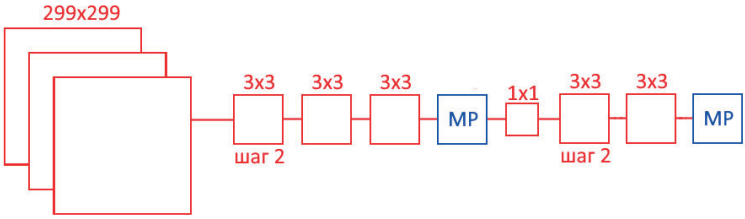
В Inception предложено некоторые архитектуры опять-таки из последовательно сверточных слоёв. В дальнейшем в архитектуре всех сверточ_ ных нейросетей её называют stem или ствол. Изобразим базово архитектуру Inception:



Ствол – это некоторый набор входящих фильтров, потом у нас идёт 9 блоков свертки A, после каждого блока A идет MaxPooling. В конце FC и Softmax по числу классов. В Inception полносвязный слой всего один, но за счёт 9 широких блоков получилось достаточно неплохая обработка информации.

С помощью ствола мы пытаемся быстро из изображения получить какие-то базовые особенности, невзирая на какие-то помехи, то есть мы считаем, что группы 3-5-7 пикселей локализованных рядом несут одно значение и пытаемся максимально эти группы по градиентам, по контурам расчленить и разобраться, что это за группа и затем уже оперировать этими группами и на них накладывать слои свертки, подвыборки. Таким образом нейросеть достаточно эффективно работает.

Теперь разберемся из чего состоит ствол Inception:



На вход подается изображение 299*299, и оно имеет 3 канала: красный синий голубой. Например, нужно найти собаку, а она может быть маленькой, собственно, для того чтобы найти маленькую собачку в разных местах изображения нам нужно локализовать, где же эта собачка находится.

Для этого применяем ствольный набор сверток, который быстро уменьшает размерность изображения, чтобы мы отыскали особенности этой собачки на фото. Первая свертка 3*3 с шагом 2, таким образом сразу уменьшаем размер изображения в два раза. Далее применяем два фильтра 3*3, поскольку они эффективней чем один фильтр 5*5. После этого применяем MaxPooling.

Затем идёт слой 1*1. Следующий за ним слой 3*3 с шагом 2, опять уменьшаем размер изображения в два раза, и далее идет слой 3*3, завершает все MaxPooling, чтобы максимально зафильтровать изображение.

На практике мы как бы «прореживаем» фильтрами изображение, после этого уже из прореженных особенностей (ключевых особенностей изображения) попытаемся разложить изображение через широкий набор фильтров – 3*3, 5*5, MaxPooling). Это напоминает функционал человеческого глаза, когда сети нейронов выделяют базовые сущности (отсутствие света или его присутствие; базовые формы), они пытаются скомпоновать их в объекты, чтобы уловить связь.

Таким образом поступаем и мы за счет слоев свертки, которые не сильно сокращают размерность изображения, но при этом они сильно взаимно перемножаются.

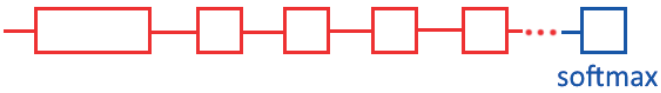
INCEPTION

Продолжая разбираться в области нейронных сетей типа inception, давайте посмотрим на основные идеи, которые развили эту архитектуру и позволили ей почти победить в очередном году на соревнованиях.

Идеи inception являются на самом деле прорывными и сейчас используются практически повсеместно в тех или иных архитектурах нейросетей.

Итак, давайте посмотрим внимательно, что это за идеи и как как они были реализованы в inception версии 3 и 4, и как мы их можем использовать для того, чтобы облегчить или улучшить точность работы нейронной сети.

Напомним, что сверточная нейронная сеть состоит из блока под названием ствол, задача которого – максимально сжать пространство признаков, и затем идут некоторые сверточные блоки (обычно одинаковые) и на выходе у нас обычно идёт полносвязный слой (или некоторый Softmax – классификатор).



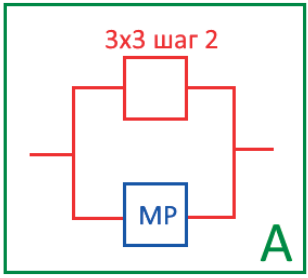
Таким образом, сверточная нейронная сеть состоит из трёх основных блоков.

Первая идея, которая касается inception: обычно в блоке ствола идет блок свертки (convolution block и блок подвыборки Max Pooling).

Свертка позволяет выделить много особенностей, но при этом сильно увеличивает пространство вариантов. Max Pooling сжимает изображение (два раза обычно), однако, при этом мы теряем некоторые характерные особенности. Поэтому возникает дилемма в выборе свертки или подвыборки.

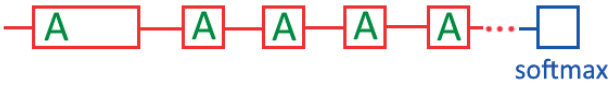
Обычно они чередуются — это одна из главных, которая легла в основу inception 3 и 4 версии. Inception предлагает использовать некоторый гибрид одновременного использования и свертки, и подвыборки.

Собственно, благодаря этому все блоки в сверточных не берите это делать и собственно все блоки в сверточных каскадах завершаются не блоком подвыборки, а некоторым гибридным блоком свертки с шагом 2 и подвыборки:



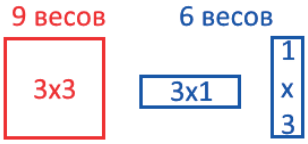
Такая комбинация позволяет получить в стволе такой характерный «блок А». То есть, мы не используем напрямую блок подвыборки, мы используем ***свертку*** с шагом 2 ***и подвыборку***. Это обогащает подвыборку, чтобы не просто отбрасывать варианты, а каким-то образом тоже на них «посмотреть» с какой-то ещё одной стороны.

Разумеется, что модифицированный «блок А» также используется во всех подвыборках, заменяя их гибридами:



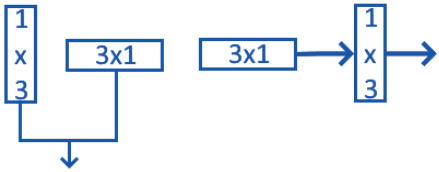
Вся суть первой идеи и состоит в том, что вместо подвыборки используется гибрид свертки + подвыборки.

Вторая идея, которая позволила сети inception стать глубже и легче заключалась в том, что вместо фильтра 3*3 будет использоваться такой вариант:



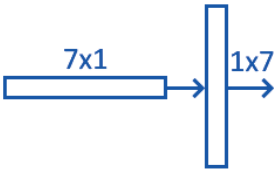
То есть фильтр 3*3 заменяется равноценным 3*1 и 1*3. То есть мы сначала выделяем горизонтальные особенности, а потом вертикальные особенности изображения. По факту, мы уже имеем всего 6 весов на каждый фильтр, вместо 9 (экономия в 1,5 раза).

Таким образом, мы ещё в полтора раза уменьшаем количество весов. Отметим также, что в блоках inception используется перестановочное перемножение, то есть можно использовать как 3*1, а потом 1*3, так и 1*3, а потом 3*1. Возможны параллельные и последовательные каскады объединения:

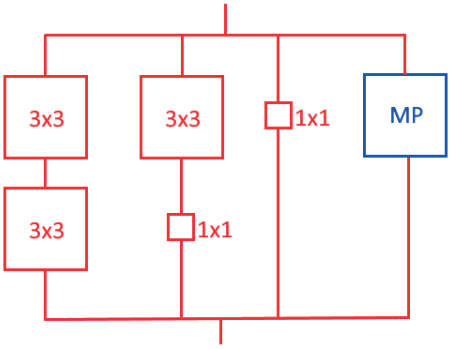


Результаты работы обеих подкаскадов в свертке объединяются для следующего сверточного слоя. Собственно, это вторая идея, которая была заложена в сеть inception – сократим количество весов за счёт того, что разобьем их на эквивалентные блоки.

Кроме блоков 1*3 и 3*1 широко распространены блоки 1*7 и 7*1, которые полностью заменяют блоки 7*7.



Кроме того, что мы таким образом облегчили фильтры и несколько расширили нашу сеть за счет параллельных сборок в каждом блоке. В общем виде сеть на примере слоя выглядит следующим образом:



Блоки могут оставаться без изменений, а могут модифицироваться на эквивалентные 3*1 / 1*3 или 7*1 / 1*7, в зависимости от того, что будет лучше в конкретной ситуации. То есть, когда мы делаем много параллельных свёрток, то предполагаем, что следующий слой нейросети разберётся, какие из этих фильтров лучше отрабатывают (тем самым, мы пытаемся «с разных сторон и под разными углами» изучить особенности данных).

И, наконец, третий момент, касающийся SoftMax – активации: было замечено, что если нейронная сеть «слишком уверена» в каком-то классе, то она перестает обучаться, возникает переобучение (наступает «подгонка» под определенный класс данных).

Но нам хотелось бы, чтобы, например, нейронная сеть, предсказывая разных птиц, выдавала некоторую вероятность конкретного примера («эта птица – воробей с вероятностью 0,67», также «эта птица – попугай с вероятностью 0,99» и «эта птица – орёл с вероятностью 0,75»). То есть, необходима ассоциативность обучения нейронной сети.

По этой причине, для того чтобы исключить жесткое переобучение нейросети, третьей важной особенностью сети inception является то, что при обучении на 1000 классов отрицательные классы (места, куда нейросеть не должна приходить при заданных входных данных) инициализировались небольшими значениями весов (порядка 0,1).

Случайная инициализация «отрицательных» классов дает нейронной сети возможность больше ошибаться, но требует от нее выдачу нескольких, наиболее вероятных ответных классов изображения. Как бы парадоксально то ни казалось, но такая методика позволяет точнее работать нейронной сети в жизни при большей аналитической ошибке, так как, например, плохо освещенный орел может быть предсказан с большей вероятностью как воробей, но при этом второе место все равно перейдет к орлу, тем самым «сдвинув» ошибку точности.

В совокупности, все эти шаги позволили существенно продвинуть успех сети Inception в наборе Image.net, поэтому с добавочными идеями от ResNet, архитектура Inception на текущий момент[4] является практически топовой.

[4] Конец ноября – начало декабря 2021 года.

RESNET

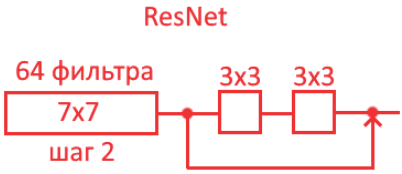
Архитектура Inception оказалась крайне успешной и в целом она обозначила некоторый путь развития сверточных нейросетей, однако на очередном соревнование по точности распознавания классов изображения была побеждена чуть более простой архитектурой ResNet.

ResNet – это так называемая остаточная нейронная сеть, то есть сеть, построенная на остатке. Архитектура является более сложной версией VGG, графически последовательность усложнения выглядит следующим образом:

LeNet ⟶ AlexNet ⟶ VGG ⟶ ResNet

Суть сети состоит не просто в том, что мы не просто расширяем в глубину нейронную сеть до 100 – 150 слоев, но при этом добавляем промежуточные связи между слоями и таким образом обучение получается легче. То есть у нас уже нет, как в случае с VGG, затухания или раскачки градиентов, у нас есть вполне понятный процесс обучения, единственное, он идёт очень долго, но зато в целом он даёт неплохую точность.

Давайте посмотрим глубже на Inception. Собственно, архитектура сверточной сети представляется стволом (ствол представляет собой 64 фильтра 7*7 с шагом 2). То есть он выполняет роль и свёртки, и подвыборки, благодаря чему мы сужаем наше изображение в два раза и получаем некоторый базовый набор особенностей из него и дальше у нас идут последовательные каскады фильтров (все фильтры 3*3), они объединены в блоки по 2 и между блоками существует перенос особенностей.

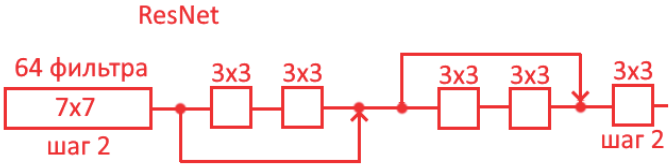


Благодаря данной стрелочке происходит перенос «остатка» между блоками. Это означает, что в одном случае происходит обработка в блоке, а в другом пропуск и передача данных к следующему слою нейронной сети (блоку), то есть существует два вида информации, протекающих от блока к блоку:

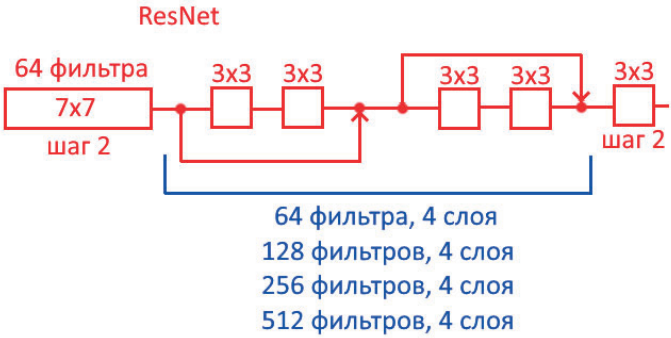
1. Обработанная информация.
2. Необработанная информация.

Это необходимо для того, чтобы нейронная сеть сама «разобралась» нужны ли ней вообще в определенных местах фильтры 3*3 или им не нужно вообще там находиться. Благодаря этому частично решается проблема с градиентом, позволяя чуть больше обучать глубокие слои. Преодолевается «раскачка» и «затухание» градиентов.

Следующий аспект: мы не будем делать слои подвыборки, а просто будем делать слои свёртки, но с шагом 2. Таким образом, свертка будет работать как подвыборка:

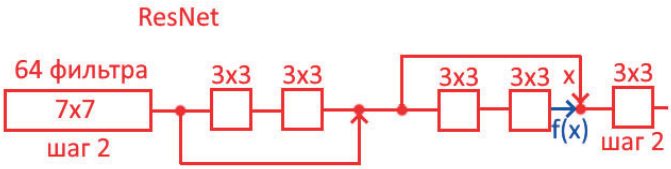


Таким образом, семейство просто наращивает глубину каждого из блоков. Сверточный каскад сохраняет 4 слоя, но фильтры в нём растут пропорционально:



То есть в наименее глубокой у нас получается ≈30 слоёв завершает всё, отказались от полносвязных слоев в конце и разрешает всё уже слой классификации после очередного слоя свертки с шагом 2 (играющего роль подвыборки).

Такая архитектура работает с затуханием градиентов при большой глубине и каждые 2 сверточных слоя происходит перенос остатков (сложение входов):



У нас есть N входов преобразованного сигнала и N входов непреобразованного сигнала, они складываются (далее с шагом 2 сверткой уменьшаются в 2 раза), благодаря этому у нас отсутствует раскачка некоторого количества ве-

сов. На каждом каскаде количество весов нейронов (количество входов нейронов) поддерживается постоянным – мы сокращаем размер изображения, увеличиваем количество фильтров, к фильтрам прибавляем «остатки» (предыдущие значения нейронов) и это всё углубляется, но число весов (число входящих связей на нейроны) остаётся константой на каждом каскаде свертки.

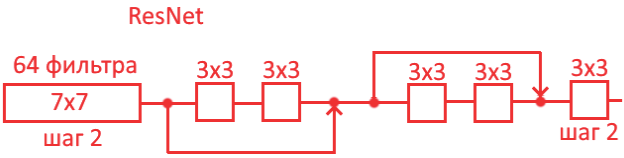
Всё семейство включает несколько сверточных нейросетей, они достаточно долго обучаются (≈60000 эпох обучения) и результирующий ансамбль включает все четыре, наиболее часто используемых вариаций ResNet.

Как минимум, это означает, что inception, как модификация некоторой архитектуры VGG, показала, как на базе чистой VGG архитектуры (слоев свертки 3*3) построить что-то еще, что может работать очень точно, при этом сохраняя возможность повысить точность за счет добавления слоев в глубину. ResNet показывает альтернативный подход по увеличению глубины нейросети, с сохранением её способности обучаться (эффективно обучаться) и с повышением её обобщающей и распознавательной способности.

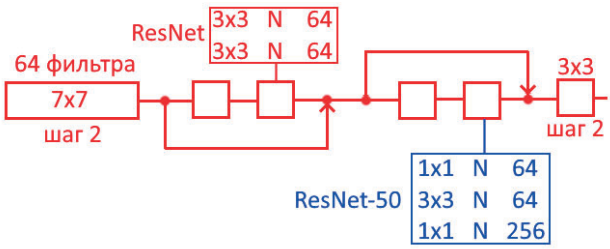
RESNETXT

Рассмотрим развитие идей ResNet, то есть можно в неё привнести зная, что у нас есть остатки, которые мы переносим между слоями, а также есть вариации блоков, которые мы можем заимствовать из GoogLeNet. Имеется в виду идея наращивание связи между слоями, так и идея расширение этих слоёв вширь.

Архитектура ResNet:

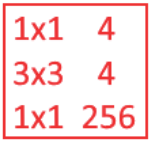


В ResNet-50:



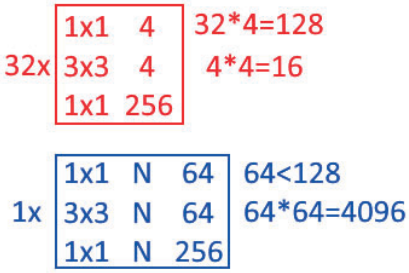
В базовой конфигурации 4 блока, объединяются в первый каскад, всего таких каскадов 4. В ResNet-50 каждый такой блок представлен одной сверткой 1*1, сверткой 3*3, дальше свертка 1*1 обладающая увеличенным количеством фильтров в четыре раза. Остальная часть архитектуры остается неизменной.

Далее рассмотрим ResNeXt. Она позволила существенно увеличить точность. Взяли идеи GoogLeNet, вместо последовательного соединения одинарных блоков развёртки, использовали несколько параллельных блоков, например:



Свертки: 1*1 4 фильтра, 3*3 4 фильтра, 1*1 256 фильтров для сохранения размерности. Количество таких маленьких блоков обозначает характеристику мощности свертки. Наилучшие результаты достигаются при мощности 32. Получается, что в ResNeXt блоки свертки представляют собой набор мелких параллельных блоков свертки.

Может возникнуть вопрос, что количеством весов в данной конфигурации, больше чем в ResNet-50:



Если взять 32 раза красные блоки, то получится что 64 < 128, но поскольку у нас идет размножение связей 64*64 = 4096, а в красном только 4*4 = 16, поэтому при обучении выходит ровно такое же количество весов.

При таком параллельном подходе, когда пытаемся выделить некоторые 32 независимые особенности из слоя изображения, чтобы их передать дальше. То есть мы не выделяем 64 фильтра 3*3, чтобы перемножить с другими вариантами этих фильтров. Считаем, что эти 32 особенности изображения будем рассматривать как независимые особенности. Чтобы именно эти 32 особенности изображения выделить, возьмём некоторые 4 выреза, наложим на них другие свертки 3*3, а далее идут вырезы 1*1 256 раз.

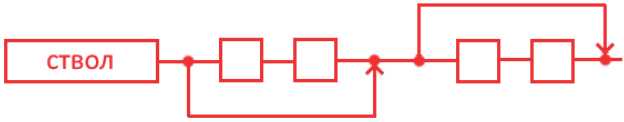
Получаем, что-то похожее на взгляд на изображение под одним конкретным углом, и мы это угол преобразуем. Эта идея очень близка к GoogLeNet, когда мы на каждом шаге накладываем несколько параллельных фильтров. В случае ResNeXt не нужны сильно разные блоки, поскольку это не добавит точности. Нам достаточно брать много одинаковых блоков.

Такая компоновка сверточной нейросети позволила ещё дальше приблизится к точности распознавания и уже стать сравнима с человеком, без увеличения нейросети в глубь. Количество фильтров на каждом слое стало больше, но они стали более разреженным, то есть мы пытаемся гармонизировать глубину шириной. Это идея нашла отражение уже в следующих поколениях нейросетей.

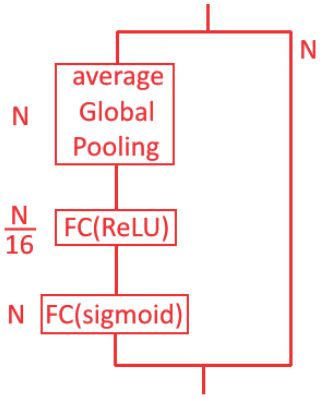
SE-RESNET

Работа с архитектурой ResNet, оказалась очень плодотворной для развития инженерной мысли и позволила реализовать несколько модификаций, которые превзошли свою прародительницу по точности. Например, модификация SE-ResNet (squeeze and excitation, сжатие и усиление).

Классическая архитектура ResNet:



В чём заключается модификация SE-ResNet? Мы не просто переносим остатки, то есть те значения, которые были на входе в подкаскад свертки, а мы их каким-то образом преобразуем. Рассмотрим это преобразование:



Оно разделяется на две части первая проходит Average Global Pooling, представляющий собой перевод пикселей из двумерного массива в одномерный. Далее идет полносвязный слой, но число входящих в него нейронов в 16 раз меньше. Затем идет sigmoid, выполняющий бинарную фильтрацию, нормировку значений, благодаря этому получаем «маску» значений. И в конце эту маску перемножаем с входящим слоем.

В итоге получаем не остатки, а некоторые значения по маске, часть из этих значений оставляем, а часть пропускаем. Это используется, когда например: на входе в каскад свертки нам нужно знать, есть ли в середине квадратов 5*5 яркие точки, то есть некоторый характерный рисунок, но он важен только на этом подкаскаде свертки. Для другого подкаскада нужен другой характерный ему рисунок.

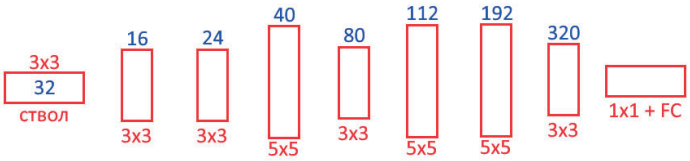
В этом и заключается идея SE-ResNet, когда используем разные маски для «подсветки» при переносе остатков. Это не сильно усложняет модель. Сложность возрастает на 1–2 %. Однако точность распознавания для топ-5 категорий архитектур, которые использовали такой перенос остатков, возросла на 1 %. А для топ-1 увеличение составило 0,5 %. Что стало еще одним шагом вперед. За несколько лет было проделано много небольших шагов, которые в результате вылились в существенный прогресс в работе с свертками в распознавании изображений.

EFFICIENTNET

Основная проблема сверточных нейросетей заключается в том, что они работают хорошо, но каждое распознавание изображения требует очень много ресурсов. Логично может быть найти чуть менее точные решения, которые будут работать на порядок быстрее. Таким подходом является EfficientNet.

Чтобы получить этот подход пришлось отыскать эталонные соотношения между сверчками и числом фильтров, которые позволили бы наращивать его, делать её глубже, шире и при этом увеличить разрешение. Получается нужно найти базовую модель, а также определиться с коэффициентами «расширения» модели.

Рассмотрим базовую модель EfficientNet-B0:



Сначала идет входящий слой свертки 3*3 – ствол. Далее идут слои свертки 3*3 и 5*5, с различным количеством фильтров (синий текст). Завер

шает некоторый полносвязный слой с сверткой 1*1. Основное преимущество – довольно малое количество фильтров, при этом сохраняется большая точность.

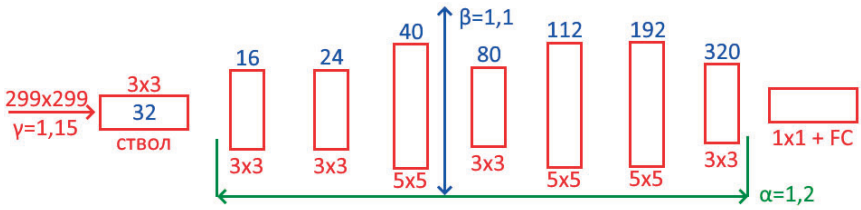
Нам важно здесь неконечное количество фильтров, а соотношение между количество фильтров, которое позволяет сбалансировать и получить некоторую очень компактную базовую сверточную нейронную сеть, которую можем расширять во все измерения. Такую идеальную архитектуру сверточной сети разработали относительно недавно. Она обладает минимумом параметров, то есть кроме хорошего распознавания изображения, она делает это с минимальным числом параметров.

Получилась самая быстрая из всех возможных классов простых сверточных нейросетей, которые не имеют дополнительных блоков, а также решают задачу распознавания изображений класса ImageNet, с заданной точностью.

EfficientNet-B0 самая быстрая и самая легкая. В конце всего 320 фильтров, в VGG, например, 512, а в некоторых 1024, 2048 и так далее.

После того как нашли методом подбора такую базовую модель и сказали теперь, что мы фиксируем некоторую вычислительную сложность по формуле:

$$\alpha * \beta^2 * \gamma^2 \approx 2.$$



α коэффициент — это некоторый множитель глубины, базовое значение 1,2. При усложнении EfficientNet, например B1 он уже $\alpha_{B1} = \alpha^2$. Также и с другими коэффициентами $\beta_{B1} = \beta^2$ и $\gamma_{B1} = \gamma^2$. Получается, что во всех формулах есть некий показатель φ, то есть степень, которая для B1 $\varphi = 2$. Для B2 $\varphi = 3$ и так далее.

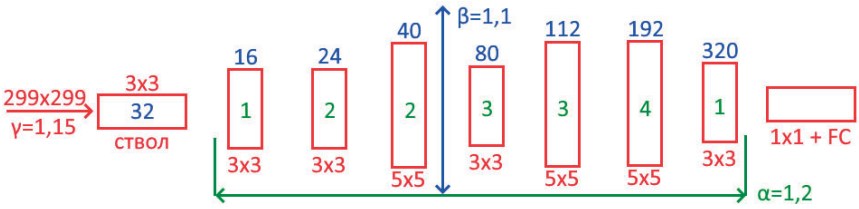
β коэффициент — это число фильтров, то есть наша ширина нейросети, базовое значение 1,1. Квадрат в формуле обусловлен тем, что увеличение количества сверток в 2 раза, увеличивает сложность нейросети в квадрат раз.

γ коэффициент показывает разрешение входящего изображения. Базовое изображение в ImageNet 299*299. А базовое значение коэффициента 1,15.

Все коэффициенты с друг другом связаны. Мы фиксируем входящее разрешение изображения и под него подбираем оптимальную архитектуру с весами и глубиной, затем мы там вычисляем γ коэффициент. Далее нам, например, нужно построить второй класс эффективной сверточной нейросети. Воз-

водим γ в квадрат, и соответственно должны взять больше фильтров и больше слоев, а также большее разрешение входящего изображения 320*320.

В этом случае мы решаем задачу минимального числа параметров, то есть количество весов, которые нам нужно обучить будет минимально, при одновременной максимизации точности работы нейросети. Рассмотрим, как нейросеть растет в глубину. Для этого используется весовые коэффициенты между различными слоями:



При наращивании слоев в первую очередь будут расти те слои, где коэффициент больше. То есть мы говорим, что если нам нужно добавить ещё один слой с $\alpha = 1{,}25$ и он пойдёт к слою с коэффициентом 4, если нам нужен ещё, то на один из слоев с коэффициентом 3. Получается, что у нас есть приоритет при добавлении слоев.

Таким образом EfficientNet позволила примерно в 10 раз меньше использовать параметров чем ResNet и DenseNet, с сохранением точности, даже с небольшим повышением точности. Это происходит из-за меньшего количества параметров и весов. Градиент меньше раскачивается, меньше затухает за тоже количество эпох обучения.

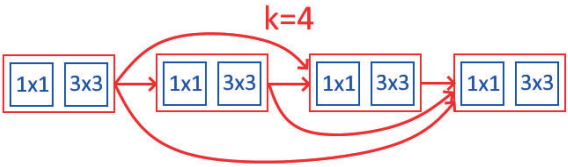
DENSENET

После успеха ResNet, а именно метода добавления некоторых дополнительных слоев остатков суммирования входящих сигналов форм, чтобы на основании сумм принимать решение о дальнейшем распознавании признаков. Решили каким-то образом облегчить архитектуру ResNet, поскольку она оказалась очень тяжелой.

На базе архитектуры ResNet была предложено архитектура DenseNet. Которая перерабатывала ее в лучшую сторону, но если ResNet это добавление остатков, то DenseNet это плотная нейросеть, другими словами, когда мы перемножаем все что можно, как бы замыкаем слои друг на друга, получаем некоторые плотные связи между признаками и пытаемся из этих плотных связей что-то извлечь.

Рассмотрим DenseNet:

В стволе есть свертка 7*7 и свертка 3*3 с шагом 2. Дальше идут так называемые плотные блоки DenseBlock. Между ними располагаются два слоя MaxPooling. После третьего DenseBlock находится AveragePooling + FC + softmax. Теперь разберемся, что такое DenseBlock:



Это блок для k = 4, k это некоторая сложность этого блока, то есть количество маленьких блоков. В рабочих версиях сети сложность равна 12 либо 24. DenseBlock представляет собой маленькие блоки состоящие из набора сверток 1*1 и 3*3. Суть DenseBlock в том, что выход из первого блока подается на вход, как второго, так и третьего, и четвертого. Выход второго блока подается на вход третьего и четвертого.

Получается у нас есть некоторое суммирование, в отличии от ResNet, где просто передавались остатки, в DenseNet плотные связи и все блоки на все последующие передают данные. В рабочих вариантах с использованием сложности 12 выходит целая «пирамида» связей. Когда в последний блок, входит не 3, а уже 11 блоков и там соответственно достаточно большой каскад параметров.

Соответственно нейросеть уже из них может выбрать, что из них является существенным, а что несущественным. У нас скажем так есть 11 вариантов обработки, 11 вариантов свертки особенностей изображения. Нейросеть их каким-то образом комбинирует, чтобы из них выделить особенности.

При сложности 24, на последний блок подается 23 варианта отношений. Это оказывается в целом достаточно, то есть мы делаем с изображением всё возможные. Можно, конечно, расширить эту идею в сторону ResNeXt, сделать некоторое распараллеливание фильтров.

Но идея DenseNet в том, что мы подаем на вход финального, который является агрегирующим сумматором, то есть он выделяет все возможные особенности из преобразований предыдущего блока. И основная его «фишка» — это то, что он может отбросить часть фильтров. При простых особенностях на изображениях, DenseBlock отбросит сложные преобразования этих простых особенностей.

Было показано, что DenseNet с глубиной 24 легче чем ResNet примерно в два раза, соответственно работает в два раза быстрее чем ResNet, еще и при увеличении точности на 1–1,5 %. По итогам очередного соревнования распознаваний изображения DenseNet уже превысил человеческую точность.

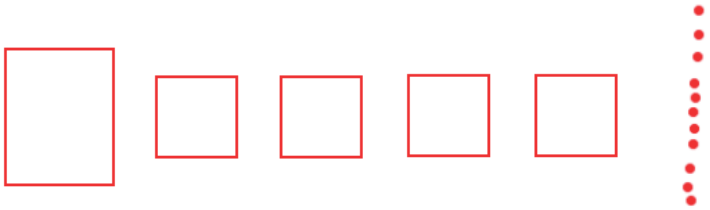
MOBILENET

После того, как сверточные нейронные сети достигли значительной точности в задачах распознавания изображений, пошла уже инженерная задача, состоящая в том, как это все можно внедрить в production (то есть прикладную область).

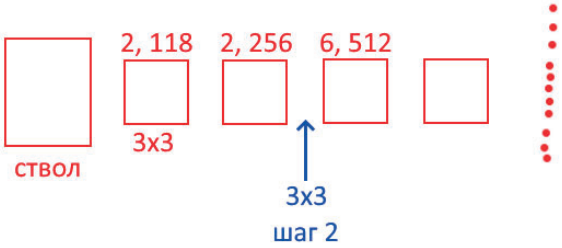
Возникла проблема расчета распознавания одной единственной фотографии, занимающая секунды. Ученые, исследователи, доктора ещё могли ожидать секунды, однако, в случае рядовых пользователей это время крайне долгое. Также следует учесть, что в мобильных платформах ограниченные ресурсы. Соответственно, от исследователей требовалось какое-то решение, которое позволило бы облегчить нейронные сети без существенных потерь в точности распознавания.

Одним из таких подходов для маломощной техники («слабых» процессоров, автономного питания – аккумулятора – накладывающего жесткие ограничения на расход процессорного времени, и так далее) стала инициатива возникновения нового семейства нейронных сетей MobileNet («mobile» – от английского «мобильный»), которая эффективно решала и продолжает решать задачу упаковки сверточной нейронной сети для мобильных устройств.

Давайте посмотрим, что из себя представляет и какие особенности хранит в себе MobileNet. За основу MobileNet была взята архитектура VGG, то есть стандартный подход, когда у нас существует некоторый ствол, на котором «схлопывается» изображение и дальше идут каскады свертки. Свертка 3*3 и далее может быть некоторый завершающий слой (полносвязный / сегментационный слой – для выделения области):

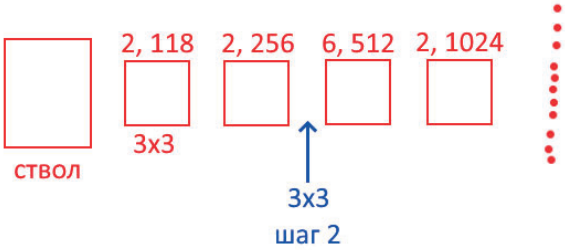


Первая особенность состоит в том, что снижение размерности в два раза достигается сверткой 3*3 с шагом в 2:



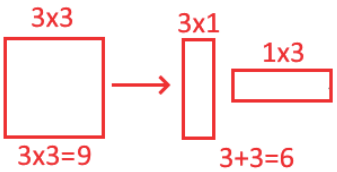
Таким образом наступает окончательное избавление от Max Pooling, поскольку это очень ресурсозатратная процедура. Вся суть замены состоит в применение свертки 3*3 с шагом в 2, которая играет роль как свертки, так и подвыборки одновременно.

В завершающем слое у нас 2 свертки по 1024 фильтра:



Получается достаточно классическая архитектура, когда мы сжимаем пространство (тем самым классически в 5 раз уменьшается размерность).

Следующая важная вещь, которая используется практически во всех MobileNet сетях – это оптимизированная свертка 3*3:

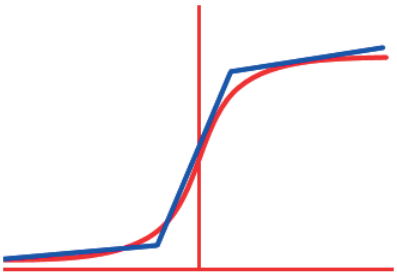


Как видно из рисунка, вместо 9 весов получается 6 практически без потери точности. Благодаря этому достигается небольшое проигрывание в точности распознавания (≈1 %), но при этом нейросеть работает в ≈10 раз быстрее. Очевидно, что баланс между точностью и производительностью в пользу скорости.

Следующим важным нюансом является то, что применяется модифицированная структура остаточных связей через SE (Squeeze and Excitation).

Напомним, что SE в стандартном виде выглядит следующим образом:

Сигмоида в блоке SE – это достаточно сложный «вычислительный камень преткновения», так как в мобильных платформах потребуется много ресурсов для такого рода вычислений. Авторы MobileNet предложили максимально приблизить сигмоиду через аппроксимирование её тремя линейными функциями:



Кусочно-линейное приближение сигмоиды, разумеется, будет работать чуть хуже, но не менее хорошо, чем такой вариант. Такой переход позволяет молниеносно отсечь ≈20–30 % от энергозатрат на потребление аккумулятора мобильными операционными системами.

В итоге, за счет того, что:

1. Отказались от Max Pooling в пользу свертки.
2. Применяем облегченные блоки свертки 3*3.
3. Облегченная сигмоида в виде аппроксимации тремя линейными функциями.

Количество весов, которое ранее равнялось ≈ 28.000.000 (по сравнению с ResNet) теперь составляет ≈ 3.000.000, что характеризует экономию практически в 10 раз.

Данные подходы позволило существенно продвинуться в распространении нейронных сетей без ухудшения точности в задачи распознавания изображения.

ЧАСТЬ ПРАКТИЧЕСКИХ НАВЫКОВ К 9

LENET И ALEXNET

Постановка задачи

Разберем архитектуру LeNet и AlexNet для решения задач распознавания изображений. Применим их для анализа исходных изображений.

Обучим модели, используя последовательную загрузку данных. Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from skimage import io
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from keras.layers import BatchNormalization, ZeroPadding2D
from keras import optimizers
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

```
Using TensorFlow backend.
```

Используемые функции

Ввод [2]:

```
filesDir = "train_images_small"
batch_size = 20
image_x = 525 # 525
image_y = 350 # 350
image_ch = 3 # 3
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
```

```
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice += 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice

def load_y (df):
    return np.array(df["EncodedPixels"].notnull().astype("int8")).reshape(len(df), 1)
```

```
def load_x (df):
    x = [[]]*len(df)
    for j, file in enumerate(df["Image"]):
        x[j] = io.imread(os.path.join(filesDir, file))
    return np.array(x).reshape(len(df), image_y, image_x, image_ch)

def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            yield (load_x(df[batch_start:limit]),
                   load_y(df[batch_start:limit]))
            batch_start += batch_size
            batch_end += batch_size
```

Загрузка данных

Ввод [3]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [4]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

```
                                          EncodedPixels         Image Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg  Fish
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg  Fish
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg  Fish
12                                                NaN  0035239.jpg  Fish
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg  Fish
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [5]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
                                                EncodedPixels         Image Label
4588                                                      NaN  3509cbc.jpg  Fish
6900   1250336 847 1251736 847 1253136 847 1254536 84...  4f4b396.jpg  Fish
7584   1162967 433 1164367 433 1165767 433 1167167 43...  567d89b.jpg  Fish
10060  1524601 714 1526001 714 1527401 714 1528801 71...  729e50e.jpg  Fish
7988   305201 1189 306601 1189 308001 1189 309401 118...  5b291dc.jpg  Fish
```

LeNet5

Первая успешная архитектура сверточной нейросети, 1998

Ввод [6]:

```
lenet = Sequential([
    Conv2D(6, (5,5), input_shape=(image_y, image_x, image_ch),
           kernel_initializer="glorot_uniform", strides=(1,1)),
    Activation("relu"),
    AveragePooling2D(pool_size=(2,2)),
    Conv2D(16, (5,5),
           kernel_initializer="glorot_uniform", strides=(1,1)),
    Activation("relu"),
    AveragePooling2D(pool_size=(2,2)),
    Flatten(),
    Activation("tanh"),
    Dense(120),
    Activation("tanh"),
    Dense(84),
    Activation("softmax"),
    Dense(1)
])
```

```
WARNING: Logging before flag parsing goes to stderr.
W0317 10:37:34.046737  1788 deprecation_wrapper.py:119] From c:\users\nikolay\appdata\local\programs\python\python37\
lib\site-packages\keras\backend\tensorflow_backend.py:3980: The name tf.nn.avg_pool is deprecated. Please use tf.nn.a
vg_pool2d instead.
```

Обучение модели

Обучим построенную модель и вычислим ее точность, используя самый лучший разделитель значений

Ввод [10]:

```
def train_evaluate_model (model):
    model.compile(optimizer=optimizers.Nadam(lr=0.05),
                  loss="mean_absolute_error")
    model.fit_generator(load_data(train, batch_size),
            epochs=50, steps_per_epoch=len(train)//batch_size)
    prediction = model.predict_generator(load_data(test, 1),
            steps=len(test), verbose=1)
    prediction = np.transpose(prediction)
    fig = plt.figure(figsize=(16,8))
    ax = fig.add_subplot(1,1,1)
    ax.hist(prediction[0])
    ax.set_title("Fish")
    plt.show()
    acc = prediction[0].mean()
    acc_max = prediction[0].max()
    if acc == acc_max:
        test["target"] = np.round(prediction[0])
        return test.apply(calc_dice, axis=1,
                          result_type="expand").mean()
    else:
        dice_best = 0
        for i in range(0,20):
            acc += (acc_max - acc)*i/20
            test["target"] = (prediction[0] >= acc).astype("int8")
            dice = test.apply(calc_dice, axis=1, result_type="expand")
            if dice_best < dice.mean():
                dice_best = dice.mean()
            else:
                break
        return dice_best
```

Ввод [11]:
```
print ("Keras, LeNet:", round(train_evaluate_model(lenet), 3))
```

```
Epoch 1/50
221/221 [==============================] - 863s 4s/step - loss: 0.4976
Epoch 2/50
221/221 [==============================] - 901s 4s/step - loss: 0.4968
Epoch 3/50
221/221 [==============================] - 882s 4s/step - loss: 0.4964
Epoch 4/50
221/221 [==============================] - 828s 4s/step - loss: 0.4968
Epoch 5/50
221/221 [==============================] - 797s 4s/step - loss: 0.4978
Epoch 6/50
221/221 [==============================] - 808s 4s/step - loss: 0.4972
Epoch 7/50
221/221 [==============================] - 791s 4s/step - loss: 0.4967
Epoch 8/50
221/221 [==============================] - 852s 4s/step - loss: 0.4971
Epoch 9/50
221/221 [==============================] - 949s 4s/step - loss: 0.4966
Epoch 10/50
```

Ввод [12]:
```
del lenet
```

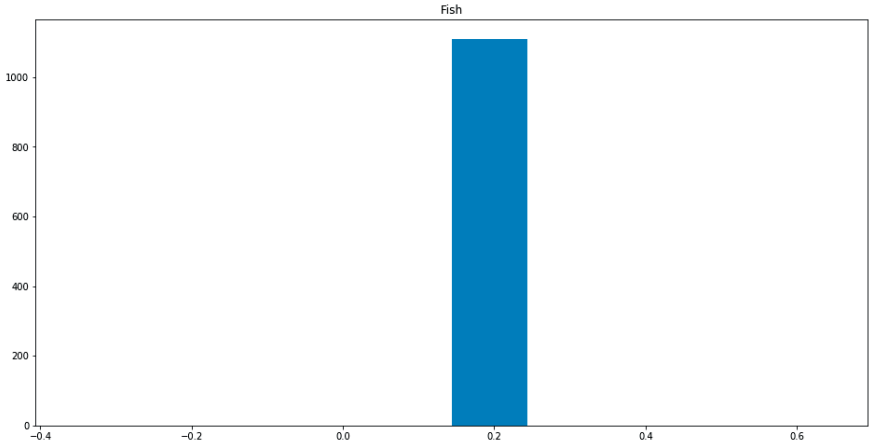
AlexNet и CaffeNet

Первая сверточная нейросеть, победившая в ImageNet
CaffeNet - однопроцессорная версия AlexNet

Для реализации потребуется задать шаг свертки (strides) и дополнительные слои для заполнения границ изображения нулями, чтобы не уменьшать область после свертки.

Ввод [15]:
```
alexnet = Sequential([
    Conv2D(96, (11,11), input_shape=(image_y, image_x, image_ch),
           kernel_initializer='glorot_uniform', strides=(4,4)),
    Activation("relu"),
    BatchNormalization(),
    MaxPooling2D(pool_size=(3,3), strides=(2,2)),
    ZeroPadding2D(padding=(2,2)),
    Conv2D(256, (5,5), kernel_initializer='glorot_uniform'),
    Activation("relu"),
    BatchNormalization(),
    MaxPooling2D(pool_size=(3,3), strides=(2,2)),
    ZeroPadding2D(padding=(1,1)),
    Conv2D(384, (3,3), kernel_initializer='glorot_uniform'),
    Activation("relu"),
    BatchNormalization(),
    ZeroPadding2D(padding=(1,1)),
    Conv2D(384, (3,3), kernel_initializer='glorot_uniform'),
    Activation("relu"),
    BatchNormalization(),
    ZeroPadding2D(padding=(1,1)),
    Conv2D(256, (3,3), kernel_initializer='glorot_uniform'),
    Activation("relu"),
    BatchNormalization(),
    MaxPooling2D(pool_size=(3,3), strides=(2,2)),
    Flatten(),
    Activation("relu"),
    Dense(1024),
    Activation("relu"),
    Dense(1024),
    Activation("softmax"),
    Dense(1)
])
```

```
Ввод [16]: print ("Keras, AlexNet:", round(train_evaluate_model(alexnet), 3))

Epoch 1/50
221/221 [==============================] - 3937s 18s/step - loss: 0.5009
Epoch 2/50
221/221 [==============================] - 4251s 19s/step - loss: 0.4972
Epoch 3/50
221/221 [==============================] - 4210s 19s/step - loss: 0.4968
Epoch 4/50
221/221 [==============================] - 5887s 27s/step - loss: 0.4966
Epoch 5/50
221/221 [==============================] - 6410s 29s/step - loss: 0.4979
Epoch 6/50
221/221 [==============================] - 9919s 45s/step - loss: 0.4984
Epoch 7/50
221/221 [==============================] - 4279s 19s/step - loss: 0.4969
Epoch 8/50
221/221 [==============================] - 15050s 68s/step - loss: 0.4970
Epoch 9/50
221/221 [==============================] - 6435s 29s/step - loss: 0.4958
Epoch 10/50
```



VGG16 И VGG19

Постановка задачи

Разберем архитектуру VGG для решения задач распознавания изображений. Применим их для анализа исходных изображений.

Используя веса обученной модели и, обучим линейный слой поверх существующей модели. Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from sklearn.model_selection import train_test_split
from keras.preprocessing import image
from keras.models import Model
from keras.layers import Dense, Flatten, Activation
from keras.applications.vgg16 import VGG16, preprocess_input
from keras import optimizers
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

```
Using TensorFlow backend.
```

Используемые функции

Ввод [2]:
```
batch_size = 20
filesDir = "train_images_small"
image_x = 224 # 525
image_y = 224 # 350
image_ch = 3 # 3
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice += 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice

def load_y (df):
    return np.array(df["EncodedPixels"].notnull().astype("int8")).reshape(len(df), 1)

def load_x (df):
    x = [[]]*len(df)
    for j, file in enumerate(df["Image"]):
        img = image.load_img(os.path.join(filesDir, file),
                             target_size=(image_y, image_x))
        img = image.img_to_array(img)
```

```
        img = np.expand_dims(img, axis=0)
        x[j] = preprocess_input(img)
    return np.array(x).reshape(len(df), image_y, image_x, image_ch)

def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            yield (load_x(df[batch_start:limit]),
                   load_y(df[batch_start:limit]))
            batch_start += batch_size
            batch_end += batch_size

def draw_prediction (prediction):
    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1,1,1)
    ax.hist(prediction[0])
    ax.set_title("Fish")
    plt.show()
```

Загрузка данных

Ввод [3]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [4]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

```
                                          EncodedPixels         Image Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg  Fish
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg  Fish
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg  Fish
12                                                NaN  0035239.jpg  Fish
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg  Fish
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [5]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
                                          EncodedPixels         Image Label
20588  227313 3 227321 9 227332 1 227335 2 227338 1 2...  ed308d4.jpg  Fish
10936                                                NaN  7c6474b.jpg  Fish
2492                                                 NaN  1bb3c7a.jpg  Fish
18864                                                NaN  da6473a.jpg  Fish
4316   396958 314 398358 314 399758 314 401158 314 40...  31226bb.jpg  Fish
```

VGG16

Используем обученную нейросеть для классификации "неизвестных" изображений - облаков

Для построения модели потребуются рассчитанные веса: 527 Мб данных.

Ввод [6]:
```
vgg16 = VGG16(weights='imagenet', include_top=False,
              input_shape=(image_y, image_x, image_ch))
```

Ввод [7]:
```
layers = Flatten()(vgg16.output)
layers = Activation("softmax")(layers)
model = Model(inputs=vgg16.input, outputs=Dense(1)(layers))
for layer in vgg16.layers:
    layer.trainable = False
model.compile(optimizer=optimizers.Nadam(lr=0.05),
              loss="mean_absolute_error")
model.summary()
```

```
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
activation_1 (Activation)    (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 25089
=================================================================
Total params: 14,739,777
Trainable params: 25,089
Non-trainable params: 14,714,688
_________________________________________________________________
```

Ввод [10]:
```
model.fit_generator(load_data(train, batch_size),
                    epochs=50, steps_per_epoch=len(train)//batch_size)
```

```
Epoch 1/10
221/221 [==============================] - 3237s 15s/step - loss: 0.4575
Epoch 2/10
221/221 [==============================] - 3191s 14s/step - loss: 0.4690
Epoch 3/10
221/221 [==============================] - 3502s 16s/step - loss: 0.4628
Epoch 4/10
221/221 [==============================] - 3380s 15s/step - loss: 0.4655
Epoch 5/10
221/221 [==============================] - 3769s 17s/step - loss: 0.4630
Epoch 6/10
221/221 [==============================] - 3340s 15s/step - loss: 0.4639
Epoch 7/10
221/221 [==============================] - 3390s 15s/step - loss: 0.4596
Epoch 8/10
221/221 [==============================] - 3610s 16s/step - loss: 0.4631
Epoch 9/10
221/221 [==============================] - 4163s 19s/step - loss: 0.4645
Epoch 10/10
221/221 [==============================] - 3658s 17s/step - loss: 0.4627
```

Out[10]: `<keras.callbacks.History at 0x1fde1948>`

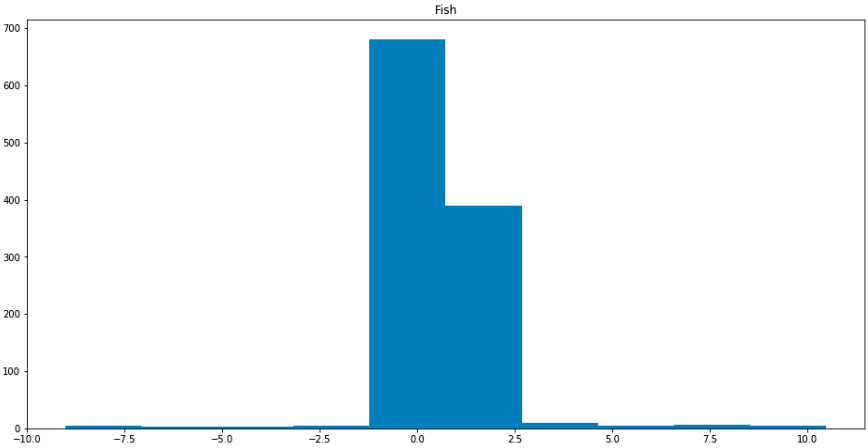
Построение предсказания

Ввод [11]:
```
prediction = model.predict_generator(load_data(test, 1),
                                      steps=len(test), verbose=1)
```

```
1110/1110 [==============================] - 1120s 1s/step
```

Ввод [12]:
```
prediction = np.transpose(prediction)
```

Ввод [13]:
```
draw_prediction(prediction)
```



Ввод [19]:
```
test["target"] = np.round(prediction[0]>5).astype("int8")
print (test[test["target"]>0][["EncodedPixels","target"]])
```

```
                                            EncodedPixels  target
18428                                                 NaN       1
13788                                                 NaN       1
3244                                                  NaN       1
8164   1286000 590 1287400 590 1288800 590 1290200 59...       1
22044  852912 515 854312 515 855712 515 857112 515 85...       1
8112                                                  NaN       1
13680  74564 440 75964 440 77364 440 78764 440 80164 ...       1
8700   719624 629 721024 629 722424 629 723824 629 72...       1
11296  25753 713 27153 713 28553 713 29953 713 31353 ...       1
3176   1 506 1401 506 2801 506 4201 506 5601 506 7001...       1
21676  650 269 2050 269 3450 269 4850 269 6250 269 76...       1
5692   34258 523 35658 523 37058 523 38458 523 39858 ...       1
11372  394116 266 394385 5 395516 260 395777 13 39691...       1
1616   2262157 238 2263557 238 2264957 238 2266357 23...       1
```

Расчет точности предсказания

Нет облаков - 0.5, MLP - 0.3, CONV - 0.48, AlexNet - 0.2

Ввод [20]:
```
dice = test.apply(calc_dice, axis=1, result_type="expand")
print ("Keras, VGG16:", round(dice.mean(), 3))
```

```
Keras, VGG16: 0.496
```

GOOGLENET И INCEPTION-BN

Постановка задачи

Разберем архитектуру GoogleLeNet для решения задач распознавания изображений. Построим эту нейросеть для анализа исходных изображений.

Обучим модели, используя последовательную загрузку данных. Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from sklearn.model_selection import train_test_split
from keras.preprocessing import image
from keras.layers import Input, Dense, Conv2D, MaxPooling2D
from keras.layers import AveragePooling2D, ZeroPadding2D, Dropout
from keras.layers import Flatten, Concatenate, Reshape, Activation
from keras.models import Model
from keras.regularizers import l2
from keras import optimizers
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

```
Using TensorFlow backend.
```

Ввод [2]:
```
keras.backend.set_image_data_format('channels_first')
```

Используемые функции

Ввод [3]:
```
filesDir = "train_images_small"
batch_size = 20
image_x = 224 # 525
image_y = 224 # 350
image_ch = 3 # 3
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice += 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice

def load_y (df):
    y = np.array(df["EncodedPixels"].notnull().astype("int8")).reshape(len(df), 1)
    return y

def load_x (df):
    x = [[]]*len(df)
```

```
    for j, file in enumerate(df["Image"]):
        img = image.load_img("train_images_small/" + file,
                             target_size=(224, 224))
        img = image.img_to_array(img)
        x[j] = np.expand_dims(img, axis=0)
    return np.array(x).reshape(len(df), image_ch, image_x, image_y)

def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            y = load_y(df[batch_start:limit])
            yield (load_x(df[batch_start:limit]),
                   [y,y,y])
            batch_start += batch_size
            batch_end += batch_size

def draw_prediction (prediction):
    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1,1,1)
    ax.hist(prediction[0])
    ax.set_title("Fish")
    plt.show()
```

Создание модели. Источник: https://gist.github.com/joelouismarino/a2ede9ab3928f999575423b9887abd14

```
def create_googlenet(weights_path=None):
    # Creates GoogLeNet a.k.a. Inception v1 (Szegedy, 2015)
    input = Input (shape= (3, 224, 224))

    input_pad = ZeroPadding2D(padding=(3, 3))(input)
    conv1_7x7_s2 = Conv2D(64, (7,7), strides=(2,2), padding='valid',
activation='relu', name='conv1/7x7_s2', kernel_regularizer=l2(0.0002))(input_pad)
    conv1_zero_pad = ZeroPadding2D(padding=(1, 1))(conv1_7x7_s2)
    pool1_helper = PoolHelper()(conv1_zero_pad)
    pool1_3x3_s2 = MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid',
name='pool1/3x3_s2')(pool1_helper)
    pool1_norm1 = LRN(name='pool1/norm1')(pool1_3x3_s2)

    conv2_3x3_reduce = Conv2D(64, (1,1), padding='same', activation='relu',
name='conv2/3x3_reduce', kernel_regularizer=l2(0.0002))(pool1_norm1)
    conv2_3x3 = Conv2D(192, (3,3), padding='same', activation='relu',
name='conv2/3x3', kernel_regularizer=l2(0.0002))(conv2_3x3_reduce)
    conv2_norm2 = LRN(name='conv2/norm2')(conv2_3x3)
    conv2_zero_pad = ZeroPadding2D(padding=(1, 1))(conv2_norm2)
    pool2_helper = PoolHelper()(conv2_zero_pad)
    pool2_3x3_s2 = MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid',
name='pool2/3x3_s2')(pool2_helper)
```

```
    inception_3a_1x1 = Conv2D(64, (1,1), padding='same', activation='relu',
name='inception_3a/1x1', kernel_regularizer=l2(0.0002))(pool2_3x3_s2)
    inception_3a_3x3_reduce = Conv2D(96, (1,1), padding='same', activation='relu',
name='inception_3a/3x3_reduce', kernel_regularizer=l2(0.0002))(pool2_3x3_s2)
    inception_3a_3x3_pad = ZeroPadding2D(padding=(1, 1))(inception_3a_3x3_
reduce)
    inception_3a_3x3 = Conv2D(128, (3,3), padding='valid', activation='relu',
name='inception_3a/3x3', kernel_regularizer=l2(0.0002))(inception_3a_3x3_pad)
    inception_3a_5x5_reduce = Conv2D(16, (1,1), padding='same', activation='relu',
name='inception_3a/5x5_reduce', kernel_regularizer=l2(0.0002))(pool2_3x3_s2)
    inception_3a_5x5_pad = ZeroPadding2D(padding=(2, 2))(inception_3a_5x5_
reduce)
    inception_3a_5x5 = Conv2D(32, (5,5), padding='valid', activation='relu',
name='inception_3a/5x5', kernel_regularizer=l2(0.0002))(inception_3a_5x5_pad)
    inception_3a_pool = MaxPooling2D(pool_size=(3,3), strides=(1,1),
padding='same', name='inception_3a/pool')(pool2_3x3_s2)
    inception_3a_pool_proj = Conv2D(32, (1,1), padding='same', activation='relu',
name='inception_3a/pool_proj', kernel_regularizer=l2(0.0002))(inception_3a_pool)
    inception_3a_output = Concatenate(axis=1, name='inception_3a/output')
([inception_3a_1x1,inception_3a_3x3,inception_3a_5x5,inception_3a_pool_proj])

    inception_3b_1x1 = Conv2D(128, (1,1), padding='same', activation='relu',
name='inception_3b/1x1', kernel_regularizer=l2(0.0002))(inception_3a_output)
    inception_3b_3x3_reduce = Conv2D(128, (1,1), padding='same', activation='relu',
name='inception_3b/3x3_reduce', kernel_regularizer=l2(0.0002))(inception_3a_
output)
    inception_3b_3x3_pad = ZeroPadding2D(padding=(1, 1))(inception_3b_3x3_
reduce)
    inception_3b_3x3 = Conv2D(192, (3,3), padding='valid', activation='relu',
name='inception_3b/3x3', kernel_regularizer=l2(0.0002))(inception_3b_3x3_pad)
    inception_3b_5x5_reduce = Conv2D(32, (1,1), padding='same', activation='relu',
name='inception_3b/5x5_reduce', kernel_regularizer=l2(0.0002))(inception_3a_
output)
    inception_3b_5x5_pad = ZeroPadding2D(padding=(2, 2))(inception_3b_5x5_
reduce)
    inception_3b_5x5 = Conv2D(96, (5,5), padding='valid', activation='relu',
name='inception_3b/5x5', kernel_regularizer=l2(0.0002))(inception_3b_5x5_pad)
    inception_3b_pool = MaxPooling2D(pool_size=(3,3), strides=(1,1),
padding='same', name='inception_3b/pool')(inception_3a_output)
    inception_3b_pool_proj = Conv2D(64, (1,1), padding='same', activation='relu',
name='inception_3b/pool_proj', kernel_regularizer=l2(0.0002))(inception_3b_pool)
    inception_3b_output = Concatenate(axis=1, name='inception_3b/output')
([inception_3b_1x1,inception_3b_3x3,inception_3b_5x5,inception_3b_pool_proj])
```

```
    inception_3b_output_zero_pad = ZeroPadding2D(padding=(1, 1))(inception_
3b_output)
    pool3_helper = PoolHelper()(inception_3b_output_zero_pad)
    pool3_3x3_s2 = MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid',
name='pool3/3x3_s2')(pool3_helper)

    inception_4a_1x1 = Conv2D(192, (1,1), padding='same', activation='relu',
name='inception_4a/1x1', kernel_regularizer=l2(0.0002))(pool3_3x3_s2)
    inception_4a_3x3_reduce = Conv2D(96, (1,1), padding='same', activation='relu',
name='inception_4a/3x3_reduce', kernel_regularizer=l2(0.0002))(pool3_3x3_s2)
    inception_4a_3x3_pad = ZeroPadding2D(padding=(1, 1))(inception_4a_3x3_
reduce)
    inception_4a_3x3 = Conv2D(208, (3,3), padding='valid', activation='relu',
name='inception_4a/3x3' ,kernel_regularizer=l2(0.0002))(inception_4a_3x3_pad)
    inception_4a_5x5_reduce = Conv2D(16, (1,1), padding='same', activation='relu',
name='inception_4a/5x5_reduce', kernel_regularizer=l2(0.0002))(pool3_3x3_s2)
    inception_4a_5x5_pad = ZeroPadding2D(padding=(2, 2))(inception_4a_5x5_
reduce)
    inception_4a_5x5 = Conv2D(48, (5,5), padding='valid', activation='relu',
name='inception_4a/5x5', kernel_regularizer=l2(0.0002))(inception_4a_5x5_pad)
    inception_4a_pool = MaxPooling2D(pool_size=(3,3), strides=(1,1),
padding='same', name='inception_4a/pool')(pool3_3x3_s2)
    inception_4a_pool_proj = Conv2D(64, (1,1), padding='same', activation='relu',
name='inception_4a/pool_proj', kernel_regularizer=l2(0.0002))(inception_4a_pool)
    inception_4a_output = Concatenate(axis=1, name='inception_4a/output')
([inception_4a_1x1,inception_4a_3x3,inception_4a_5x5,inception_4a_pool_proj])

    loss1_ave_pool = AveragePooling2D(pool_size=(5,5), strides=(3,3),
name='loss1/ave_pool')(inception_4a_output)
    loss1_conv = Conv2D(128, (1,1), padding='same', activation='relu', name='loss1/
conv', kernel_regularizer=l2(0.0002))(loss1_ave_pool)
    loss1_flat = Flatten()(loss1_conv)
    loss1_fc = Dense(1024, activation='relu', name='loss1/fc',
kernel_regularizer=l2(0.0002))(loss1_flat)
    loss1_drop_fc = Dropout(rate=0.7)(loss1_fc)
    loss1_classifier_act = Activation('softmax')(loss1_drop_fc)
    loss1_classifier_final = Dense(1, kernel_regularizer=l2(0.0002))(loss1_
classifier_act)

    inception_4b_1x1 = Conv2D(160, (1,1), padding='same', activation='relu',
name='inception_4b/1x1', kernel_regularizer=l2(0.0002))(inception_4a_output)
    inception_4b_3x3_reduce = Conv2D(112, (1,1), padding='same',
activation='relu', name='inception_4b/3x3_reduce',
kernel_regularizer=l2(0.0002))(inception_4a_output)
```

```
    inception_4b_3x3_pad = ZeroPadding2D(padding=(1, 1))(inception_4b_3x3_
reduce)
    inception_4b_3x3 = Conv2D(224, (3,3), padding='valid', activation='relu',
name='inception_4b/3x3', kernel_regularizer=l2(0.0002))(inception_4b_3x3_pad)
    inception_4b_5x5_reduce = Conv2D(24, (1,1), padding='same',
activation='relu', name='inception_4b/5x5_reduce',
kernel_regularizer=l2(0.0002))(inception_4a_output)
    inception_4b_5x5_pad = ZeroPadding2D(padding=(2, 2))(inception_4b_5x5_
reduce)
    inception_4b_5x5 = Conv2D(64, (5,5), padding='valid', activation='relu',
name='inception_4b/5x5', kernel_regularizer=l2(0.0002))(inception_4b_5x5_pad)
    inception_4b_pool = MaxPooling2D(pool_size=(3,3), strides=(1,1),
padding='same', name='inception_4b/pool')(inception_4a_output)
    inception_4b_pool_proj = Conv2D(64, (1,1), padding='same', activation='relu',
name='inception_4b/pool_proj', kernel_regularizer=l2(0.0002))(inception_4b_pool)
    inception_4b_output = Concatenate(axis=1, name='inception_4b/output')
([inception_4b_1x1,inception_4b_3x3,inception_4b_5x5,inception_4b_pool_proj])

    inception_4c_1x1 = Conv2D(128, (1,1), padding='same', activation='relu',
name='inception_4c/1x1', kernel_regularizer=l2(0.0002))(inception_4b_output)
    inception_4c_3x3_reduce = Conv2D(128, (1,1), padding='same',
activation='relu', name='inception_4c/3x3_reduce',
kernel_regularizer=l2(0.0002))(inception_4b_output)
    inception_4c_3x3_pad = ZeroPadding2D(padding=(1, 1))(inception_4c_3x3_
reduce)
    inception_4c_3x3 = Conv2D(256, (3,3), padding='valid', activation='relu',
name='inception_4c/3x3', kernel_regularizer=l2(0.0002))(inception_4c_3x3_pad)
    inception_4c_5x5_reduce = Conv2D(24, (1,1), padding='same', activation='relu',
name='inception_4c/5x5_reduce',
kernel_regularizer=l2(0.0002))(inception_4b_output)
    inception_4c_5x5_pad = ZeroPadding2D(padding=(2, 2))(inception_4c_5x5_
reduce)
    inception_4c_5x5 = Conv2D(64, (5,5), padding='valid', activation='relu',
name='inception_4c/5x5', kernel_regularizer=l2(0.0002))(inception_4c_5x5_pad)
    inception_4c_pool = MaxPooling2D(pool_size=(3,3), strides=(1,1),
padding='same', name='inception_4c/pool')(inception_4b_output)
    inception_4c_pool_proj = Conv2D(64, (1,1), padding='same', activation='relu',
name='inception_4c/pool_proj', kernel_regularizer=l2(0.0002))(inception_4c_pool)
    inception_4c_output = Concatenate(axis=1, name='inception_4c/output')
([inception_4c_1x1,inception_4c_3x3,inception_4c_5x5,inception_4c_pool_proj])

    inception_4d_1x1 = Conv2D(112, (1,1), padding='same', activation='relu',
name='inception_4d/1x1', kernel_regularizer=l2(0.0002))(inception_4c_output)
```

```
    inception_4d_3x3_reduce = Conv2D(144, (1,1), padding='same', activation='relu',
name='inception_4d/3x3_reduce',
kernel_regularizer=l2(0.0002))(inception_4c_output)
    inception_4d_3x3_pad = ZeroPadding2D(padding=(1, 1))(inception_4d_3x3_
reduce)
    inception_4d_3x3 = Conv2D(288, (3,3), padding='valid', activation='relu',
name='inception_4d/3x3', kernel_regularizer=l2(0.0002))(inception_4d_3x3_pad)
    inception_4d_5x5_reduce = Conv2D(32, (1,1), padding='same',
activation='relu', name='inception_4d/5x5_reduce',
kernel_regularizer=l2(0.0002))(inception_4c_output)
    inception_4d_5x5_pad = ZeroPadding2D(padding=(2, 2))(inception_4d_5x5_
reduce)
    inception_4d_5x5 = Conv2D(64, (5,5), padding='valid', activation='relu',
name='inception_4d/5x5', kernel_regularizer=l2(0.0002))(inception_4d_5x5_pad)
    inception_4d_pool = MaxPooling2D(pool_size=(3,3), strides=(1,1),
padding='same', name='inception_4d/pool')(inception_4c_output)
    inception_4d_pool_proj = Conv2D(64, (1,1), padding='same', activation='relu',
name='inception_4d/pool_proj', kernel_regularizer=l2(0.0002))(inception_4d_pool)
    inception_4d_output = Concatenate(axis=1, name='inception_4d/output')
([inception_4d_1x1,inception_4d_3x3,inception_4d_5x5,inception_4d_pool_proj])

    loss2_ave_pool = AveragePooling2D(pool_size=(5,5), strides=(3,3),
name='loss2/ave_pool')(inception_4d_output)
    loss2_conv = Conv2D(128, (1,1), padding='same', activation='relu',
name='loss2/conv', kernel_regularizer=l2(0.0002))(loss2_ave_pool)
    loss2_flat = Flatten()(loss2_conv)
    loss2_fc = Dense(1024, activation='relu', name='loss2/fc', kernel_regularizer=l2
(0.0002))(loss2_flat)
    loss2_drop_fc = Dropout(rate=0.7)(loss2_fc)
    loss2_classifier_act = Activation('softmax')(loss2_drop_fc)
    loss2_classifier_final = Dense(1,
kernel_regularizer=l2(0.0002))(loss2_classifier_act)

    inception_4e_1x1 = Conv2D(256, (1,1), padding='same', activation='relu',
name='inception_4e/1x1', kernel_regularizer=l2(0.0002))(inception_4d_output)
    inception_4e_3x3_reduce = Conv2D(160, (1,1), padding='same', activation='relu',
name='inception_4e/3x3_reduce',
kernel_regularizer=l2(0.0002))(inception_4d_output)
    inception_4e_3x3_pad = ZeroPadding2D(padding=(1, 1))(inception_4e_3x3_
reduce)
    inception_4e_3x3 = Conv2D(320, (3,3), padding='valid', activation='relu',
name='inception_4e/3x3', kernel_regularizer=l2(0.0002))(inception_4e_3x3_pad)
```

```
    inception_4e_5x5_reduce = Conv2D(32, (1,1), padding='same', activation='relu',
name='inception_4e/5x5_reduce', kernel_regularizer=l2(0.0002))(inception_4d_
output)
    inception_4e_5x5_pad = ZeroPadding2D(padding=(2, 2))(inception_4e_5x5_
reduce)
    inception_4e_5x5 = Conv2D(128, (5,5), padding='valid', activation='relu',
name='inception_4e/5x5', kernel_regularizer=l2(0.0002))(inception_4e_5x5_pad)
    inception_4e_pool = MaxPooling2D(pool_size=(3,3), strides=(1,1),
padding='same', name='inception_4e/pool')(inception_4d_output)
    inception_4e_pool_proj = Conv2D(128, (1,1), padding='same', activation='relu',
name='inception_4e/pool_proj', kernel_regularizer=l2(0.0002))(inception_4e_pool)
    inception_4e_output = Concatenate(axis=1, name='inception_4e/output')
([inception_4e_1x1,inception_4e_3x3,inception_4e_5x5,inception_4e_pool_proj])

    inception_4e_output_zero_pad = ZeroPadding2D(padding=(1, 1))(inception_
4e_output)
    pool4_helper = PoolHelper()(inception_4e_output_zero_pad)
    pool4_3x3_s2 = MaxPooling2D(pool_size=(3,3), strides=(2,2), padding='valid',
name='pool4/3x3_s2')(pool4_helper)

    inception_5a_1x1 = Conv2D(256, (1,1), padding='same', activation='relu',
name='inception_5a/1x1', kernel_regularizer=l2(0.0002))(pool4_3x3_s2)
    inception_5a_3x3_reduce = Conv2D(160, (1,1), padding='same', activation='relu',
name='inception_5a/3x3_reduce', kernel_regularizer=l2(0.0002))(pool4_3x3_s2)
    inception_5a_3x3_pad = ZeroPadding2D(padding=(1, 1))(inception_5a_3x3_
reduce)
    inception_5a_3x3 = Conv2D(320, (3,3), padding='valid', activation='relu',
name='inception_5a/3x3', kernel_regularizer=l2(0.0002))(inception_5a_3x3_pad)
    inception_5a_5x5_reduce = Conv2D(32, (1,1), padding='same', activation='relu',
name='inception_5a/5x5_reduce', kernel_regularizer=l2(0.0002))(pool4_3x3_s2)
    inception_5a_5x5_pad = ZeroPadding2D(padding=(2, 2))(inception_5a_5x5_
reduce)
    inception_5a_5x5 = Conv2D(128, (5,5), padding='valid', activation='relu',
name='inception_5a/5x5', kernel_regularizer=l2(0.0002))(inception_5a_5x5_pad)
    inception_5a_pool = MaxPooling2D(pool_size=(3,3), strides=(1,1),
padding='same', name='inception_5a/pool')(pool4_3x3_s2)
    inception_5a_pool_proj = Conv2D(128, (1,1), padding='same', activation='relu',
name='inception_5a/pool_proj', kernel_regularizer=l2(0.0002))(inception_5a_pool)
    inception_5a_output = Concatenate(axis=1, name='inception_5a/output')
([inception_5a_1x1,inception_5a_3x3,inception_5a_5x5,inception_5a_pool_proj])

    inception_5b_1x1 = Conv2D(384, (1,1), padding='same', activation='relu',
name='inception_5b/1x1', kernel_regularizer=l2(0.0002))(inception_5a_output)
```

```
    inception_5b_3x3_reduce = Conv2D(192, (1,1), padding='same', activation='relu',
name='inception_5b/3x3_reduce', kernel_regularizer=l2(0.0002))(inception_5a_
output)
    inception_5b_3x3_pad = ZeroPadding2D(padding=(1, 1))(inception_5b_3x3_
reduce)
    inception_5b_3x3 = Conv2D(384, (3,3), padding='valid', activation='relu',
name='inception_5b/3x3', kernel_regularizer=l2(0.0002))(inception_5b_3x3_pad)
    inception_5b_5x5_reduce = Conv2D(48, (1,1), padding='same', activation='relu',
name='inception_5b/5x5_reduce',
kernel_regularizer=l2(0.0002))(inception_5a_output)
    inception_5b_5x5_pad = ZeroPadding2D(padding=(2, 2))(inception_5b_5x5_
reduce)
    inception_5b_5x5 = Conv2D(128, (5,5), padding='valid', activation='relu',
name='inception_5b/5x5', kernel_regularizer=l2(0.0002))(inception_5b_5x5_pad)
    inception_5b_pool = MaxPooling2D(pool_size=(3,3), strides=(1,1),
padding='same', name='inception_5b/pool')(inception_5a_output)
    inception_5b_pool_proj = Conv2D(128, (1,1), padding='same', activation='relu',
name='inception_5b/pool_proj', kernel_regularizer=l2(0.0002))(inception_5b_pool)
    inception_5b_output = Concatenate(axis=1, name='inception_5b/output')
([inception_5b_1x1,inception_5b_3x3,inception_5b_5x5,inception_5b_pool_proj])

    pool5_7x7_s1 = AveragePooling2D(pool_size=(7,7), strides=(1,1),
name='pool5/7x7_s2')(inception_5b_output)
    loss3_flat = Flatten()(pool5_7x7_s1)
    pool5_drop_7x7_s1 = Dropout(rate=0.4)(loss3_flat)
    loss3_classifier_act = Activation('softmax', name='prob')(pool5_drop_7x7_s1)
    loss3_classifier_final = Dense(1, kernel_regularizer=l2(0.0002))(loss3_
classifier_act)

    googlenet = Model(inputs=input,
        outputs=[loss1_classifier_final, loss2_classifier_final,
                loss3_classifier_final])

    if weights_path:
        googlenet.load_weights(weights_path)

    return googlenet
```

Ввод [6]:
```
from keras.layers.core import Layer
from keras import backend as K
if K.backend() == 'theano':
    import theano.tensor as T
elif K.backend() == 'tensorflow':
    import tensorflow as tf
else:
    raise NotImplementedError

class LRN(Layer):
    def __init__(self, alpha=0.0001, k=1, beta=0.75, n=5, **kwargs):
        self.alpha = alpha
        self.k = k
        self.beta = beta
        self.n = n
        super(LRN, self).__init__(**kwargs)
    def call(self, x, mask=None):
        b, ch, r, c = x.shape
        half_n = self.n // 2 # half the local region
        input_sqr = K.square(x) # square the input
        if K.backend() == 'theano':
            # make an empty tensor with zero pads along channel dimension
            zeros = T.alloc(0., b, ch + 2*half_n, r, c)
            # set the center to be the squared input
            input_sqr = T.set_subtensor(zeros[:, half_n:half_n+ch, :, :], input_sqr)
        else:
            input_sqr = tf.pad(input_sqr, [[0, 0], [half_n, half_n], [0, 0], [0, 0]])
        scale = self.k # offset for the scale
        norm_alpha = self.alpha / self.n # normalized alpha
        for i in range(self.n):
            scale += norm_alpha * input_sqr[:, i:i+ch, :, :]
        scale = scale ** self.beta
        x = x / scale
        return x
    def get_config(self):
        config = {"alpha": self.alpha,
                  "k": self.k,
                  "beta": self.beta,
                  "n": self.n}
        base_config = super(LRN, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))
```

Ввод [7]:
```
from keras.layers.core import Layer
class PoolHelper(Layer):
    def __init__(self, **kwargs):
        super(PoolHelper, self).__init__(**kwargs)
    def call(self, x, mask=None):
        return x[:,:,1:,1:]
    def get_config(self):
        config = {}
        base_config = super(PoolHelper, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))
```

Загрузка данных

Ввод [8]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [9]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [10]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
                                                EncodedPixels         Image Label
3252   686887 513 688287 513 689687 513 691087 513 69...  2485e01.jpg  Fish
9128                                                     NaN  68373b5.jpg  Fish
1636   529237 248 530637 248 532037 248 533437 248 53...  12e10fa.jpg  Fish
9984                                                     NaN  71ecd2d.jpg  Fish
19548                                                    NaN  e1b35d8.jpg  Fish
```

GoogLeNet (Inception v1)

Обучим нейросеть на уменьшенных изображениях

Ввод [12]:
```
googlenet = create_googlenet()
googlenet.compile(optimizer=optimizers.Nadam(lr=0.05),
                  loss="mean_absolute_error")
googlenet.summary()
```

```
dropout_6 (Dropout)             (None, 1024)         0           flatten_6[0][0]
activation_3 (Activation)       (None, 1024)         0           dropout_4[0][0]
activation_4 (Activation)       (None, 1024)         0           dropout_5[0][0]
prob (Activation)               (None, 1024)         0           dropout_6[0][0]
dense_4 (Dense)                 (None, 1)            1025        activation_3[0][0]
dense_5 (Dense)                 (None, 1)            1025        activation_4[0][0]
dense_6 (Dense)                 (None, 1)            1025        prob[0][0]
==================================================================================================
Total params: 10,306,355
Trainable params: 10,306,355
Non-trainable params: 0
```

Ввод [14]:
```
googlenet.fit_generator(load_data(train, batch_size),
                        epochs=20, steps_per_epoch=len(train)//batch_size)
```

```
Epoch 1/20
221/221 [==============================] - 8130s 37s/step - loss: 10.9142 - dense_4_loss: 0.5084 - dense_5_loss: 0.51
54 - dense_6_loss: 0.5147
Epoch 2/20
221/221 [==============================] - 4693s 21s/step - loss: 10.2544 - dense_4_loss: 0.5189 - dense_5_loss: 0.51
76 - dense_6_loss: 0.5235
Epoch 3/20
221/221 [==============================] - 23265s 105s/step - loss: 9.8321 - dense_4_loss: 0.5292 - dense_5_loss: 0.5
231 - dense_6_loss: 0.5099
Epoch 4/20
221/221 [==============================] - 2798s 13s/step - loss: 9.5569 - dense_4_loss: 0.5185 - dense_5_loss: 0.513
0 - dense_6_loss: 0.5275
Epoch 5/20
221/221 [==============================] - 2791s 13s/step - loss: 9.3615 - dense_4_loss: 0.5236 - dense_5_loss: 0.511
9 - dense_6_loss: 0.5172
Epoch 6/20
221/221 [==============================] - 2794s 13s/step - loss: 9.2439 - dense_4_loss: 0.5166 - dense_5_loss: 0.510
4 - dense_6_loss: 0.5253
Epoch 7/20
221/221 [==============================] - 2873s 13s/step - loss: 9.0966 - dense_4_loss: 0.5164 - dense_5_loss: 0.518
6 - dense_6_loss: 0.5058
Epoch 8/20
221/221 [==============================] - 2799s 13s/step - loss: 8.9829 - dense_4_loss: 0.5165 - dense_5_loss: 0.511
0 - dense_6_loss: 0.5170
Epoch 9/20
221/221 [==============================] - 2797s 13s/step - loss: 8.8587 - dense_4_loss: 0.5171 - dense_5_loss: 0.507
7 - dense_6_loss: 0.5084
Epoch 10/20
221/221 [==============================] - 2813s 13s/step - loss: 8.7528 - dense_4_loss: 0.5093 - dense_5_loss: 0.508
2 - dense_6_loss: 0.5054
Epoch 11/20
221/221 [==============================] - 2836s 13s/step - loss: 8.6780 - dense_4_loss: 0.5074 - dense_5_loss: 0.510
1 - dense_6_loss: 0.5087
Epoch 12/20
221/221 [==============================] - 2806s 13s/step - loss: 8.5889 - dense_4_loss: 0.5056 - dense_5_loss: 0.507
6 - dense_6_loss: 0.5176
Epoch 13/20
221/221 [==============================] - 2795s 13s/step - loss: 8.5015 - dense_4_loss: 0.5126 - dense_5_loss: 0.509
2 - dense_6_loss: 0.5040
```

```
Epoch 14/20
221/221 [==============================] - 2791s 13s/step - loss: 8.4162 - dense_4_loss: 0.5079 - dense_5_loss: 0.509
9 - dense_6_loss: 0.5066
Epoch 15/20
221/221 [==============================] - 2790s 13s/step - loss: 8.3481 - dense_4_loss: 0.5064 - dense_5_loss: 0.507
2 - dense_6_loss: 0.5075
Epoch 16/20
221/221 [==============================] - 2791s 13s/step - loss: 8.2645 - dense_4_loss: 0.5073 - dense_5_loss: 0.510
0 - dense_6_loss: 0.5041
Epoch 17/20
221/221 [==============================] - 2796s 13s/step - loss: 8.1953 - dense_4_loss: 0.5140 - dense_5_loss: 0.506
1 - dense_6_loss: 0.5050
Epoch 18/20
221/221 [==============================] - 2797s 13s/step - loss: 8.1292 - dense_4_loss: 0.5107 - dense_5_loss: 0.504
5 - dense_6_loss: 0.5046
Epoch 19/20
221/221 [==============================] - 2808s 13s/step - loss: 8.0490 - dense_4_loss: 0.5053 - dense_5_loss: 0.501
8 - dense_6_loss: 0.5080
Epoch 20/20
221/221 [==============================] - 3015s 14s/step - loss: 7.9782 - dense_4_loss: 0.5042 - dense_5_loss: 0.498
0 - dense_6_loss: 0.5037
```

Out[14]: <keras.callbacks.History at 0x34c77a48>

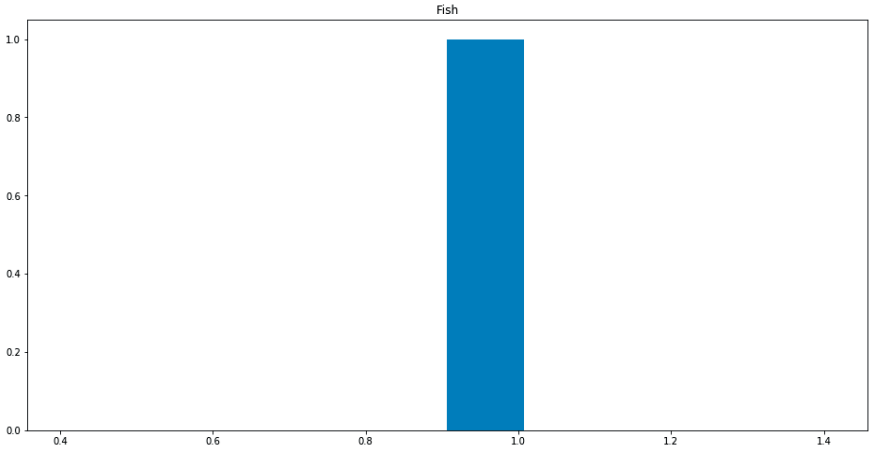
Построение предсказания

Ввод [15]:
```
prediction = googlenet.predict_generator(load_data(test, 1),
                                         steps=len(test), verbose=1)
```

```
1110/1110 [==============================] - 300s 270ms/step
```

Ввод [22]:
```
print (np.array(prediction[2]).mean())
draw_prediction(prediction[2])
```

```
0.9068998
```



Ввод [23]:
```
test["target"] = round(np.array(prediction[2]).mean())
print (test.head())
```

```
                                        EncodedPixels        Image Label  \
10196                                             NaN  746443b.jpg  Fish
16568  77642 429 79042 429 80442 429 81842 429 83242 ...  bf28fa6.jpg  Fish
8056                                              NaN  5be3705.jpg  Fish
3952   1064001 943 1064945 16 1065401 940 1066345 13 ...  2d2c35f.jpg  Fish
16108                                             NaN  b8f9a9d.jpg  Fish

       target
10196     1.0
16568     1.0
8056      1.0
3952      1.0
16108     1.0
```

Расчет точности предсказания

Нет облаков - 0.5, MLP - 0.298, CONV/VGG - 0.48, AlexNet - 0.2

Ввод [24]:
```
dice = test.apply(calc_dice, axis=1, result_type="expand")
print ("Keras, GoogLeNet:", round(dice.mean(), 3))
```

```
Keras, GoogLeNet: 0.213
```

INCEPTION V3 И V4

Постановка задачи

Разберем архитектуру Inception для решения задач распознавания изображений. Построим эту нейросеть для анализа исходных изображений.

Используя обученную модель, построим предсказания. Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [9]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from skimage import io
from sklearn.model_selection import train_test_split
from keras.preprocessing import image
from keras.models import Model, Sequential
from keras.layers import Dense, Flatten, Activation
from keras.layers import BatchNormalization, Dropout
from keras.applications.inception_v3 import InceptionV3
from keras import optimizers
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

Используемые функции

Ввод [2]:
```
filesDir = "train_images_small"
batch_size = 50
image_x = 299 # 525
image_y = 299 # 350
image_ch = 3 # 3
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice += 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice

def load_y (df):
    return np.array(df["EncodedPixels"].notnull().astype("int8")).reshape(len(df), 1)

def load_x (df):
    x = [[]]*len(df)
    for j, file in enumerate(df["Image"]):
        img = image.load_img(os.path.join(filesDir, file),
                             target_size=(image_y, image_x))
        img = image.img_to_array(img)
        x[j] = np.expand_dims(img, axis=0)
    return np.array(x).reshape(len(df), image_y, image_x, image_ch)

def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            yield (load_x(df[batch_start:limit]),
                   load_y(df[batch_start:limit]))
            batch_start += batch_size
            batch_end += batch_size

def draw_prediction (prediction):
    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1,1,1)
    ax.hist(prediction[0])
    ax.set_title("Fish")
    plt.show()
```

Загрузка данных

Ввод [3]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [4]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

```
                                              EncodedPixels        Image Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg  Fish
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg  Fish
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg  Fish
12                                                NaN  0035239.jpg  Fish
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg  Fish
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [5]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
                                              EncodedPixels        Image Label
8572                                                    NaN  619232f.jpg  Fish
20308  1043050 308 1043359 4 1043364 4 1043376 17 104...  ea17711.jpg  Fish
9800                                                    NaN  6ef9413.jpg  Fish
17104                                                   NaN  c55412b.jpg  Fish
1716   11463 733 12863 733 14263 733 15663 733 17063 ...  136f59b.jpg  Fish
```

Inception v3

Подключим обученную нейросеть (89 Мб) и построим поверх классификатора новые слои. Используем результат работы обученной нейросети как входной слой для обучения последнего слоя, нашего классификатора.

Ввод [6]:
```
inc_model = InceptionV3(weights='imagenet', include_top=False,
                        input_shape=(image_y, image_X, image_ch))
```

```
WARNING: Logging before flag parsing goes to stderr.
W0325 12:05:10.745032  2724 deprecation_wrapper.py:119] From c:\users\nikolay\appdata\local\programs\python\python37\
lib\site-packages\keras\backend\tensorflow_backend.py:1834: The name tf.nn.fused_batch_norm is deprecated. Please use
tf.compat.v1.nn.fused_batch_norm instead.

W0325 12:05:12.887155  2724 deprecation_wrapper.py:119] From c:\users\nikolay\appdata\local\programs\python\python37\
lib\site-packages\keras\backend\tensorflow_backend.py:3980: The name tf.nn.avg_pool is deprecated. Please use tf.nn.a
vg_pool2d instead.
```

Ввод [7]:
```
inc_model.compile(optimizer="sgd", loss="mean_absolute_error")
inc_model_output = inc_model.predict_generator(load_data(train, 1),
                        steps=len(train), verbose=1)
```

```
4436/4436 [==============================] - 3404s 767ms/step
```

Ввод [12]:
```
top_model = Sequential([
    Flatten(input_shape=inc_model_output.shape[1:]),
    BatchNormalization(),
    Dropout(0.5),
    Activation("softmax"),
    Dense(1)
])
top_model.compile(optimizer=optimizers.Nadam(lr=0.02),
                  loss="mean_absolute_error")
```

Обучение модели

Ввод [13]:
```
top_model.fit(inc_model_output, load_y(train), epochs=100)
```

```
Epoch 92/100
4436/4436 [==============================] - 63s 14ms/step - loss: 0.4195
Epoch 93/100
4436/4436 [==============================] - 63s 14ms/step - loss: 0.4215
Epoch 94/100
4436/4436 [==============================] - 63s 14ms/step - loss: 0.4147
Epoch 95/100
4436/4436 [==============================] - 63s 14ms/step - loss: 0.4172
Epoch 96/100
4436/4436 [==============================] - 63s 14ms/step - loss: 0.4115
Epoch 97/100
4436/4436 [==============================] - 63s 14ms/step - loss: 0.4110
Epoch 98/100
4436/4436 [==============================] - 63s 14ms/step - loss: 0.4108
Epoch 99/100
4436/4436 [==============================] - 63s 14ms/step - loss: 0.4034
Epoch 100/100
4436/4436 [==============================] - 63s 14ms/step - loss: 0.4118
```

Out[13]:
```
<keras.callbacks.History at 0x39aaa688>
```

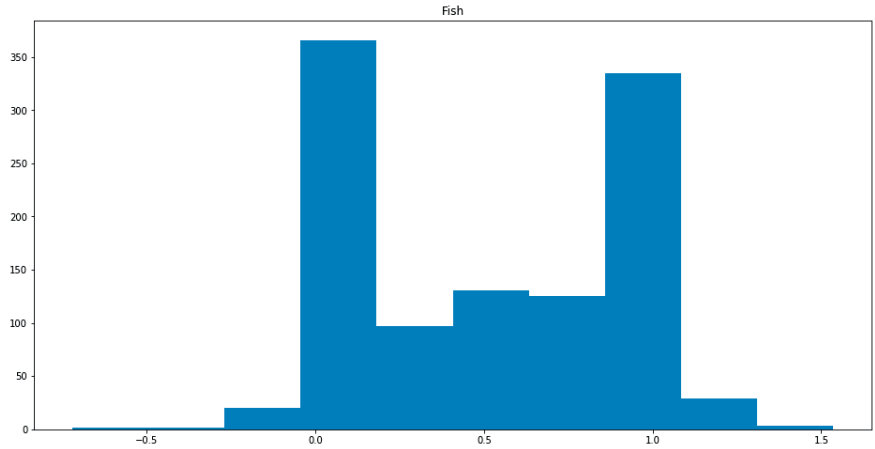
Построение предсказания

Ввод [15]:
```
prediction = model.predict_generator(load_data(test, 1),
                                     steps=len(test), verbose=1)
```

```
1110/1110 [==============================] - 752s 678ms/step
```

Ввод [16]:
```
prediction = np.transpose(prediction)
```

Ввод [17]:
```
draw_prediction(prediction)
```



Ввод [29]:
```
test["target"] = (prediction[0]>=1.1).astype("int8")
print (test[test["target"]>0][["EncodedPixels","target"]])
```

```
                                                EncodedPixels  target
22032  24699 501 26099 501 27499 501 28899 501 30299 ...       1
2116   59303 487 60703 487 62103 487 63503 487 64903 ...       1
18288  51802 628 53202 628 54602 628 56002 628 57402 ...       1
15824                                                 NaN       1
13144  873 353 2273 353 3673 353 5073 353 6473 353 78...       1
19940  20167 462 21567 462 22967 462 24367 462 25767 ...       1
14016                                                 NaN       1
16068                                                 NaN       1
11036                                                 NaN       1
8668   7406 982 8806 982 10206 982 11606 982 13006 98...       1
19440                                                 NaN       1
10148                                                 NaN       1
2716   460433 8 460457 16 461833 8 461857 16 463233 8...       1
15888                                                 NaN       1
9548   17422 221 18822 221 20222 221 21622 221 23022 ...       1
19856  451185 480 452585 480 453985 480 455385 480 45...       1
12896  45620 3 45638 2 45642 2 45655 1 45658 1 45660 ...       1
7376   39489 633 40889 633 42289 633 43689 633 45089 ...       1
```

Расчет точности предсказания

Нет облаков - 0.5, MLP - 0.3, CONV/VGG16 - 0.48, AlexNex - 0.21

Ввод [30]:
```
dice = test.apply(calc_dice, axis=1, result_type="expand")
print ("Keras, Inception v3:", round(dice.mean(), 3))
```

```
Keras, Inception v3: 0.496
```

RESNET

Постановка задачи

Разберем архитектуру ResNet для решения задач распознавания изображений. Построим эту нейросеть для анализа исходных изображений.

Используя обученную модель, построим предсказания и классификацию, используя LightGBM. Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from skimage import io
from sklearn.model_selection import train_test_split
from keras.preprocessing import image
from keras.models import Model, Sequential
from keras.layers import Dense, GlobalAveragePooling2D, Activation, GlobalMaxPooling2D
from keras.applications.resnet50 import ResNet50,preprocess_input, decode_predictions
import lightgbm as lgb
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

```
Using TensorFlow backend.
```

Используемые функции

Ввод [2]:
```
filesDir = "train_images_small"
batch_size = 20
image_x = 224 # 525
image_y = 224 # 350
image_ch = 3 # 3
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice += 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice

def load_y (df):
    return np.array(df["EncodedPixels"].notnull().astype("int8")).reshape(len(df), 1)

def load_x (df):
    x = [[]]*len(df)
    for j, file in enumerate(df["Image"]):
        img = image.load_img(os.path.join(filesDir, file),
                             target_size=(image_y, image_x))
        img = image.img_to_array(img)
        x[j] = np.expand_dims(img, axis=0)
    return np.array(x).reshape(len(df), image_y, image_x, image_ch)

def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            yield (load_x(df[batch_start:limit]),
                   load_y(df[batch_start:limit]))
            batch_start += batch_size
            batch_end += batch_size

def draw_prediction (prediction):
    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1,1,1)
    ax.hist(prediction[0])
    ax.set_title("Fish")
    plt.show()
```

Загрузка данных

Ввод [3]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [4]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

```
                                          EncodedPixels        Image Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg  Fish
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg  Fish
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg  Fish
12                                                NaN  0035239.jpg  Fish
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg  Fish
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [5]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
      EncodedPixels        Image Label
21784           NaN  fafc316.jpg  Fish
6076            NaN  4623c6c.jpg  Fish
8624            NaN  624f5eb.jpg  Fish
16760           NaN  c200cce.jpg  Fish
7392            NaN  548cb15.jpg  Fish
```

ResNet-50

Подключим обученную нейросеть (90 Мб) и построим поверх классификатора LightGBM.

Ввод [6]:
```
model = ResNet50(weights="imagenet")
```

```
WARNING: Logging before flag parsing goes to stderr.
W0325 23:38:42.359072 11888 deprecation_wrapper.py:119] From c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\keras\backend\tensorflow_backend.py:4185: The name tf.truncated_normal is deprecated. Please use tf.random.truncated_normal instead.

W0325 23:38:42.456077 11888 deprecation_wrapper.py:119] From c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\keras\backend\tensorflow_backend.py:1834: The name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.
```

Ввод [7]:
```
train_prediction = model.predict_generator(load_data(train, 1),
                        steps=len(train), verbose=1)
```

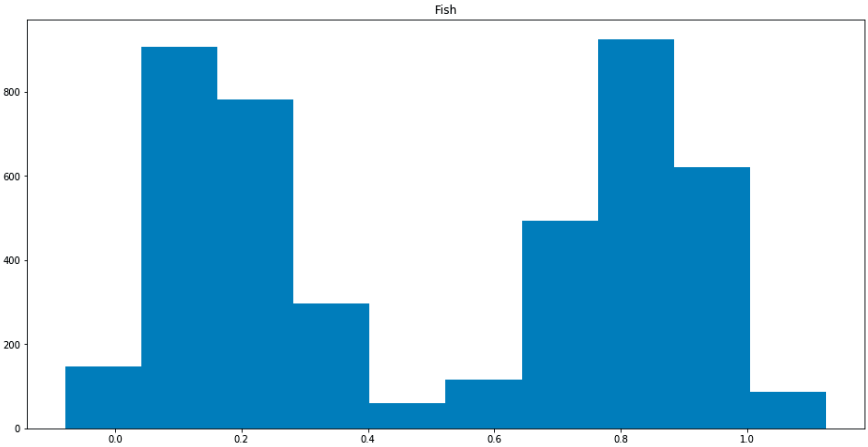
```
4436/4436 [==============================] - 3257s 734ms/step
```

Обучение LightGBM по результатам ResNet

Ввод [9]:
```
model_lgb = lgb.LGBMRegressor(random_state=17)
model_lgb.fit(pd.DataFrame(train_prediction),
              train["EncodedPixels"].notnull().astype("i1"))
```

Out[9]:
```
LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
              importance_type='split', learning_rate=0.1, max_depth=-1,
              min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
              n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
              random_state=17, reg_alpha=0.0, reg_lambda=0.0, silent=True,
              subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

```
Ввод [10]: prediction = model_lgb.predict(train_prediction)

Ввод [11]: prediction = np.array(prediction).reshape(1, -1)

Ввод [12]: draw_prediction (prediction)
```




```
Ввод [14]: train["target"] = (prediction[0] > 0.75).astype("i1")
           print (train[train["target"]>0]["EncodedPixels"])
```

```
18032    217001 491 218401 491 219801 491 221201 491 22...
16864    1582201 906 1583601 906 1585001 906 1586401 90...
12776    1095518 646 1096918 646 1098318 646 1099718 64...
2372     14010 398 15410 398 16810 398 18210 398 19610 ...
12996    200647 533 202047 533 203447 533 204847 533 20...
1432     87 809 1487 809 2887 809 4287 809 5687 809 708...
1056     548972 709 550372 709 551772 709 553172 709 55...
18680    103729 652 105129 652 106529 652 107929 652 10...
9476     966001 874 967401 874 968801 874 970201 874 97...
11116    20371 472 21771 472 23171 472 24571 472 25971 ...
19524    214947 587 216347 587 217747 587 219147 587 22...
8952     141 567 1541 567 2941 567 4341 567 5741 567 71...
7812     256 983 1656 983 3056 983 4456 983 5856 983 72...
20756    1528356 438 1529756 438 1531156 438 1532556 43...
3144     505401 216 506521 16 506541 4 506546 2 506549 ...
12916    9036 431 10436 431 11836 431 13236 431 14636 4...
14932    1050022 369 1050394 4 1050401 4 1051422 369 10...
15428    1 840 1401 840 2801 840 4201 840 5601 840 7001...
8352     318128 327 319528 327 320928 327 322328 327 32...
```

```
Ввод [15]: dice = train.apply(calc_dice, axis=1, result_type="expand")
           print ("Keras, Resnet-50+LightGBM, обучение:", round(dice.mean(), 3))
```

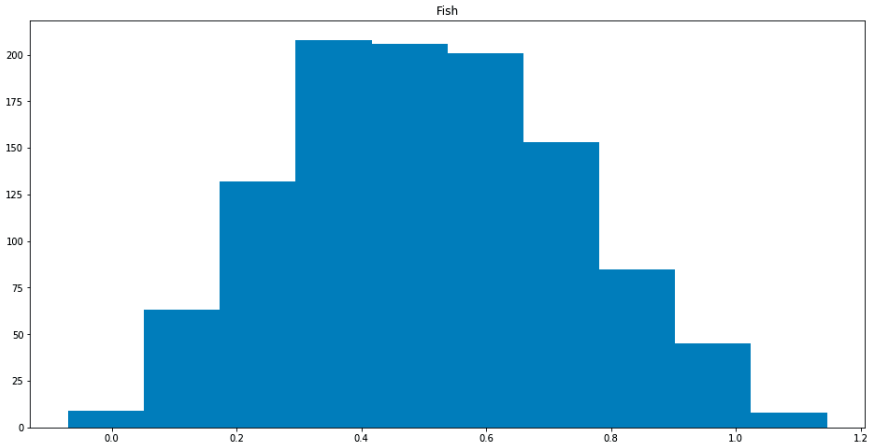
```
Keras, Resnet-50+LightGBM, обучение: 0.669
```

Построение предсказания

```
Ввод [16]: test_prediction = model.predict_generator(load_data(test, 1),
                                steps=len(test), verbose=1)
```

```
1110/1110 [==============================] - 733s 661ms/step
```

```
Ввод [10]: prediction = model_lgb.predict(train_prediction)

Ввод [11]: prediction = np.array(prediction).reshape(1, -1)

Ввод [12]: draw_prediction (prediction)
```




```
Ввод [20]: test["target"] = (prediction[0] > 0.75).astype("i1")
```

Расчет точности предсказания

Нет облаков - 0.5, MLP - 0.3, CONV/VGG - 0.48, AlexNet - 0.21, Inception - 0.5

```
Ввод [21]: dice = test.apply(calc_dice, axis=1, result_type="expand")
           print ("Keras, ResNet-50+LightGBM:", round(dice.mean(), 3))

Keras, ResNet-50+LightGBM: 0.553
```

АРХИТЕКТУРА НЕЙРОСЕТИ

Постановка задачи

Разработаем оптимальную архитектуру нейронной сети, исходя из исходной задачи классификации.

Разделим данные на обучающие и проверочные в соотношении 80/20.

Используем Keras для построения нейросети с линейным, сверточными слоями и слоями подвыборки.

Проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [9]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from sklearn.model_selection import train_test_split
from skimage import io
from keras.preprocessing import image
from keras.models import Model
from keras.layers import Input, Dense, Activation, Flatten
from keras.layers import Concatenate, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D, Dropout, ZeroPadding2D
from keras.regularizers import l2
from keras.callbacks import EarlyStopping
from keras import optimizers
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

Используемые функции

Ввод [2]:

```
filesDir = "train_images_small"
batch_size = 20
image_x = 525
image_y = 350
image_ch = 3
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return np.round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y) - 1:mask_rate(l+i, x, y)] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.ones(image_x*image_y, dtype='uint8')
        dice += 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice

def load_y (df):
    return np.array(df["EncodedPixels"].notnull().astype("int8")).reshape(len(df), 1)

def load_x (df):
    x = [[]]*len(df)
```

```
    for j, file in enumerate(df["Image"]):
        img = image.load_img(os.path.join(filesDir, file),
                             target_size=(image_y, image_x))
        img = image.img_to_array(img)
        x[j] = np.expand_dims(img, axis=0)
    return np.array(x).reshape(len(df), image_y, image_x, image_ch)

def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            yield (load_x(df[batch_start:limit]),
                   load_y(df[batch_start:limit]))
            batch_start += batch_size
            batch_end += batch_size

def draw_prediction (prediction):
    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1,1,1)
    ax.hist(prediction[0])
    ax.set_title("Fish")
    plt.show()
```

Загрузка данных

Ввод [4]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [5]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

```
                                          EncodedPixels        Image Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg  Fish
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg  Fish
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg  Fish
12                                                NaN  0035239.jpg  Fish
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg  Fish
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [6]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
                                          EncodedPixels        Image Label
13216                                               NaN  965b308.jpg  Fish
7092   1288267 1121 1289667 1121 1291067 1121 1292467...  5122a3d.jpg  Fish
10220                                               NaN  74808ec.jpg  Fish
14052  57604 446 59004 446 60404 446 61804 446 63204 ...  a0e6825.jpg  Fish
3160                                                NaN  2385676.jpg  Fish
```

Для выделения характерных особенностей на исходном изображении достаточно ввести несколько сверточных блоков, как предложено в GoogLeNet/Inception: 1×1, 3×3, 2-3×3 и объединить их через MaxPool.

Количество таких блоков будет зависеть от размера области на изображении, которую классифицируем. В случае изображения 525×350 «пятно» облака занимает 2/3 по каждому измерению, до 350×233.

Перед запуском сверточных блоков добавим инициализацию: свертку 3×3 и подвыборку, чтобы уменьшить размерность изображения, это сократит «пятно» в 4 раза, до 58 пикселей. После этого каждый сверточно-выборочный блок уменьшает размер «пятна» в 2 раза => необходимо 3–4 таких блока для успешной классификации (минимальный размер блока после финальной подвыборки – 4, lg(58/4) = 3,8).

Padding-слои нужны для выравнивания размеров входов для объединения на выходе нескольких потоков обработки изображения.

Архитектура нейросети

Ввод [40]:

```
inp = Input(shape=(image_y, image_x, image_ch))
start_3x3 = Conv2D(32, (3,3), padding="valid",
                   strides=(2,2),
                   kernel_regularizer=l2(0.0002),
                   kernel_initializer='glorot_uniform')(inp)
start_3x3_act = Activation("relu")(start_3x3)
start_3x3_pool = MaxPooling2D(pool_size=(2,2),
                    padding="same")(start_3x3_act)
input_ = start_3x3_pool
for i in range(3):
    inc_1x1 = Conv2D(32, (1,1), padding="valid",
                   kernel_regularizer=l2(0.0002),
                   kernel_initializer='glorot_uniform')(input_)
    inc_1x1_act = Activation("relu")(inc_1x1)
    inc_3x3 = Conv2D(16, (1,1), padding="valid",
                   kernel_regularizer=l2(0.0002),
                   kernel_initializer='glorot_uniform')(input_)
    inc_3x3_act = Activation("relu")(inc_3x3)
    inc_3x3_2 = Conv2D(32, (3,3), padding="valid",
                   kernel_regularizer=l2(0.0002),
                   kernel_initializer='glorot_uniform')(inc_3x3_act)
    inc_3x3_2_pad = ZeroPadding2D(padding=(1,1))(inc_3x3_2)
    inc_3x3_2_act = Activation("relu")(inc_3x3_2_pad)
    inc_5x5 = Conv2D(16, (1,1), padding="valid",
                   kernel_regularizer=l2(0.0002),
                   kernel_initializer='glorot_uniform')(input_)
    inc_5x5_act = Activation("relu")(inc_5x5)
    inc_5x5_2 = Conv2D(32, (5,5), padding="valid",
                   kernel_regularizer=l2(0.0002),
                   kernel_initializer='glorot_uniform')(inc_5x5_act)
    inc_5x5_2_pad = ZeroPadding2D(padding=(2,2))(inc_5x5_2)
    inc_5x5_2_act = Activation("relu")(inc_5x5_2_pad)
    inc_pool = MaxPooling2D(pool_size=(3,3), strides=(1,1),
                    padding="same")(input_)
    inc_pool_1x1 = Conv2D(32, (1,1), padding="same",
                   kernel_regularizer=l2(0.0002),
                   kernel_initializer='glorot_uniform')(inc_pool)
    inc_pool_1x1_act = Activation("relu")(inc_pool_1x1)
    inc_conc = Concatenate(axis=1)([inc_1x1_act, inc_3x3_2_act,
                    inc_5x5_2_act, inc_pool_1x1_act])
```

```
    inc_output = MaxPooling2D(pool_size=(2,2),
                              padding="same")(inc_conc)
    input_ = inc_output
flat = Flatten()(input_)
flat_bn = BatchNormalization()(flat)
drop = Dropout(0.5)(flat_bn)
drop_act = Activation("softmax")(drop)
final = Dense(1)(drop_act)
model = Model(inputs=inp, outputs=final)
```

Ввод [46]:
```
model.compile(optimizer=optimizers.Nadam(lr=0.001),
              loss="mean_absolute_error")
model.summary()
```

```
                                                                 activation_221[0][0]
                                                                 activation_222[0][0]
__________________________________________________________________________________________________
max_pooling2d_81 (MaxPooling2D) (None, 696, 17, 32)  0           concatenate_33[0][0]
__________________________________________________________________________________________________
flatten_10 (Flatten)            (None, 378624)       0           max_pooling2d_81[0][0]
__________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 378624)       1514496     flatten_10[0][0]
__________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 378624)       0           batch_normalization_9[0][0]
__________________________________________________________________________________________________
activation_223 (Activation)     (None, 378624)       0           dropout_9[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 1)            378625      activation_223[0][0]
==================================================================================================
Total params: 1,955,937
Trainable params: 1,198,689
Non-trainable params: 757,248
__________________________________________________________________________________________________
```

Проверка модели

Будем проверять эффективность модели на выборке 100 элементов и 10 эпохах обучения. Эффективная модель будет разделять предсказываемые классы облаков (распределение предсказания будет бимодальным).

Добавим раннюю остановку обучения, если качество предсказания не улучшается в течение 5 эпох (patience).

Ввод [47]:
```
model.fit_generator(load_data(train, batch_size),
            epochs=100, steps_per_epoch=len(train)//batch_size,
            verbose=True,
            callbacks=[EarlyStopping(monitor="loss",
                min_delta=0.0001, patience=5, verbose=1, mode="auto")])
```

```
Epoch 12/100
221/221 [==============================] - 1580s 7s/step - loss: 0.4470
Epoch 13/100
221/221 [==============================] - 1550s 7s/step - loss: 0.4410
Epoch 14/100
221/221 [==============================] - 1542s 7s/step - loss: 0.4374
Epoch 15/100
221/221 [==============================] - 1544s 7s/step - loss: 0.4403
Epoch 16/100
221/221 [==============================] - 1551s 7s/step - loss: 0.4600
Epoch 17/100
221/221 [==============================] - 1549s 7s/step - loss: 0.4547
Epoch 18/100
221/221 [==============================] - 1547s 7s/step - loss: 0.4505
Epoch 19/100
221/221 [==============================] - 1549s 7s/step - loss: 0.4479
Epoch 00019: early stopping
```

Out[47]:
```
<keras.callbacks.History at 0x6b312888>
```

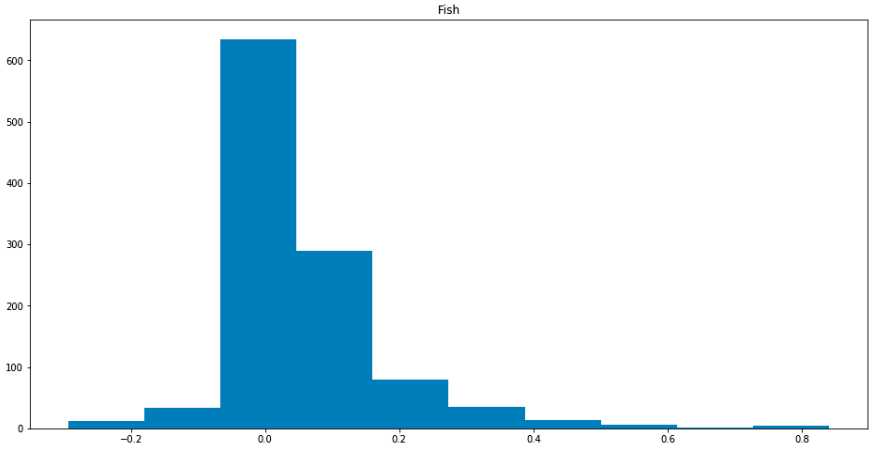
Необходимо отметить, что модели понадобилось всего 19 эпох, вместо предполагаемых (с существенным запасом) 100.

Ранняя остановка позволила получить следующие значения точности:

Ввод [48]:
```
prediction = model.predict_generator(load_data(test, 1),
                                     steps=len(test), verbose=1)
```

```
1110/1110 [==============================] - 165s 149ms/step
```

Ввод [49]:
```
prediction = np.transpose(prediction)
draw_prediction(prediction)
```



Предсказание значений

Ввод [56]:
```
test["target"] = (prediction[0] >= 0.5).astype("i1")
```

Ввод [57]:
```
print (test[test["target"]>0]["EncodedPixels"].head(100))
```

```
11996                                                  NaN
4716     1043235 786 1044635 786 1046035 786 1047435 78...
13692    304082 904 305482 904 306882 904 308282 904 30...
4808     830320 891 831720 891 833120 891 834520 891 83...
1816                                                   NaN
1964     586621 1173 588021 1173 589421 1173 590821 117...
17236                                                  NaN
16488    681915 838 683315 838 684715 838 686115 838 68...
17668                                                  NaN
11164    705617 8 707017 8 708402 7 708417 8 709802 5 7...
9228     339667 482 341067 482 342467 482 343867 482 34...
Name: EncodedPixels, dtype: object
```

Оценка по Дайсу

Пока будем считать, что при определении типа облака на изображении, оно целиком размещено на фотографии: т.е. область облака - это все изображение.

Нет облаков - 0.5, MLP - 0.3, CONV/VGG - 0.48, AlexNet - 0.2, Inception - 0.5, ResNet - 0.55

Ввод [58]:
```
dice = test.apply(calc_dice, axis=1, result_type="expand")
print ("Keras, (CONV3-32x2,POOL2,CONV1/CONV3/CONV5-x3,POOL2):",
       round(dice.mean(), 3))
```

```
Keras, (CONV3-32x2,POOL2,CONV1/CONV3/CONV5-x3,POOL2) 0.486
```

MOBILENET ДЛЯ РАЗЛИЧНЫХ ПРЕДМЕТНЫХ ОБЛАСТЕЙ

Постановка задачи

Разберем архитектуру MobileNet и проведем частичное обучение, экспорт и импорт модели, затем дообучение.

Перейдем от задачи классификации изображения к задаче локализации объекта на изображении при помощи якорей.

Построим предсказание по обученной нейросети и проведем оценку качества предсказания по коэффициенту сходства.

Данные:

1. https://video.ittensive.com/machine-learning/clouds/train.csv.gz (54 Мб)
2. https://video.ittensive.com/machine-learning/clouds/train_images_small.tar.gz (212 Мб)

Подключение библиотек

Ввод [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import keras
from sklearn.model_selection import train_test_split
from keras.preprocessing import image
from keras.models import Sequential, Model
from keras.layers import Dense, Flatten, Activation
from keras.layers import Conv2D, BatchNormalization, Dropout
from keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input
from keras.regularizers import l2
from keras import optimizers
from keras.callbacks import EarlyStopping, ModelCheckpoint
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

```
Using TensorFlow backend.
```

Используемые функции

Разобьем изображение на 5×5 квадратов и будем предсказывать наличие облаков заданной формы в каждом из квадратов.

При вычислении коэффициента Дайса «растянем» предсказанную маску до размеров изображения.

Ввод [2]:
```
batch_size = 10
filesDir = "train_images_small"
image_x = 525 # 525
image_y = 350 # 350
image_ch = 3 # 3
mask_x = 5
mask_y = 5
def mask_rate (a, x, y):
    b = a//1400 + 0.0
    return round(x*(b*x//2100) + y*(a%1400)//1400).astype("uint32")

def calc_mask (px, x=image_x, y=image_y):
    p = np.array([int(n) for n in px.split(' ')]).reshape(-1,2)
    mask = np.zeros(x*y, dtype='uint8')
    for i, l in p:
        mask[mask_rate(i, x, y):mask_rate(l+i, x, y)+1] = 1
    return mask.reshape(y,x).transpose()

def calc_dice (x):
    dice = 0
    px = x["EncodedPixels"]
    if px != px and x["target"] == 0:
        dice = 1
    elif px == px and x["target"] == 1:
        mask = calc_mask(px).flatten()
        target = np.kron(np.array(x["MaskPixels"].split(" ")).reshape(mask_x,
            mask_y).astype("i1"),
            np.ones((image_y//mask_y, image_x//mask_x),
            dtype="i1")).transpose().flatten()
        dice = 2*np.sum(target[mask==1])/(np.sum(target)+np.sum(mask))
    return dice

def load_y (df):
    y = [[0]]*len(df)
    for i, ep in enumerate(df["EncodedPixels"]):
        if ep == ep:
            y[i] = calc_mask(ep, mask_x, mask_y).transpose().flatten()
        else:
            y[i] = np.zeros(mask_x*mask_y, dtype="i1")
    return np.array(y).reshape(len(df), mask_y, mask_x, 1)

def load_x (df):
    x = [[]]*len(df)
    for j, file in enumerate(df["Image"]):
        img = image.load_img(os.path.join(filesDir, file),
                             target_size=(image_y, image_x))
        img = image.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        x[j] = preprocess_input(img)
    return np.array(x).reshape(len(df), image_y, image_x, image_ch)

def load_data (df, batch_size):
    while True:
        batch_start = 0
        batch_end = batch_size
        while batch_start < len(df):
            limit = min(batch_end, len(df))
            yield (load_x(df[batch_start:limit]),
                   load_y(df[batch_start:limit]))
            batch_start += batch_size
            batch_end += batch_size

def draw_prediction (prediction):
    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1,1,1)
    ax.hist(prediction[0])
    ax.set_title("Fish")
    plt.show()
```

Загрузка данных

Ввод [3]:
```
data = pd.read_csv('https://video.ittensive.com/machine-learning/clouds/train.csv.gz')
```

Ввод [4]:
```
data["Image"] = data["Image_Label"].str.split("_").str[0]
data["Label"] = data["Image_Label"].str.split("_").str[1]
data.drop(labels=["Image_Label"], axis=1, inplace=True)
data_fish = data[data["Label"] == "Fish"]
print (data_fish.head())
```

```
                                          EncodedPixels         Image Label
0   264918 937 266318 937 267718 937 269118 937 27...  0011165.jpg  Fish
4   233813 878 235213 878 236613 878 238010 881 23...  002be4f.jpg  Fish
8   3510 690 4910 690 6310 690 7710 690 9110 690 1...  0031ae9.jpg  Fish
12                                                NaN  0035239.jpg  Fish
16  2367966 18 2367985 2 2367993 8 2368002 62 2369...  003994e.jpg  Fish
```

Разделение данных

Разделим всю выборку на 2 части случайным образом: 80% - для обучения модели, 20% - для проверки точности модели.

Ввод [5]:
```
train, test = train_test_split(data_fish, test_size=0.2)
train = pd.DataFrame(train)
test = pd.DataFrame(test)
del data
print (train.head())
```

```
                                          EncodedPixels         Image Label
336                                                 NaN  0493d31.jpg  Fish
19640                                               NaN  e2e4fa9.jpg  Fish
11728                                               NaN  85e5f09.jpg  Fish
21896  744166 585 745566 585 746966 585 748366 585 74...  fcb25d2.jpg  Fish
13636  376601 279 378001 279 379401 279 380801 279 38...  9b25c0d.jpg  Fish
```

MobileNetV2

Используем обученную нейросеть и bn/dropout/softmax-слой поверх. Обучим модель из последнего слоя 50 эпох, сохраним обученные данные, затем загрузим их и продолжим обучение.

Используем раннюю остановку и сохранение модели после каждой эпохи обучения.

Ввод [6]:
```
base_model = MobileNetV2(weights='imagenet', include_top=False,
                         input_shape=(image_y, image_x, image_ch))
base_model_output = base_model.predict_generator(load_data(train, 1),
                        steps=len(train), verbose=1)
top_model = Sequential([
    Flatten(input_shape=base_model_output.shape[1:]),
    BatchNormalization(),
    Dropout(0.5),
    Activation("softmax"),
    Dense(mask_x * mask_y)
])
top_model.compile(optimizer=optimizers.Nadam(lr=0.05),
                  loss="mean_absolute_error")
```

```
c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\keras_applications\mobilenet_v2.py:294: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
  warnings.warn('`input_shape` is undefined or non-square, '
WARNING: Logging before flag parsing goes to stderr.
W0329 14:40:28.480491  8572 deprecation_wrapper.py:119] From c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\keras\backend\tensorflow_backend.py:1834: The name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.

4436/4436 [==============================] - 5813s 1s/step
W0329 16:21:53.125513  8572 deprecation.py:506] From c:\users\nikolay\appdata\local\programs\python\python37\lib\site-packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

Ввод [7]:
```
top_model.fit(base_model_output,
              load_y(train).reshape(len(train), -1), epochs=100,
              callbacks=[EarlyStopping(monitor="loss",
                 min_delta=0.0001, patience=5, verbose=1, mode="auto"),
                 ModelCheckpoint("mobilenet.clouds.h5", mode="auto",
                 monitor="val_loss", verbose=1)])
```

```
Epoch 00016: saving model to mobilenet.clouds.h5
Epoch 17/100
4436/4436 [==============================] - 136s 31ms/step - loss: 0.3008

Epoch 00017: saving model to mobilenet.clouds.h5
Epoch 18/100
4436/4436 [==============================] - 131s 30ms/step - loss: 0.2991

Epoch 00018: saving model to mobilenet.clouds.h5
Epoch 19/100
4436/4436 [==============================] - 134s 30ms/step - loss: 0.3009

Epoch 00019: saving model to mobilenet.clouds.h5
Epoch 20/100
4436/4436 [==============================] - 135s 30ms/step - loss: 0.2991

Epoch 00020: saving model to mobilenet.clouds.h5
Epoch 00020: early stopping
```

Out[7]:
```
<keras.callbacks.History at 0x56fba88>
```

Продолжение обучения

Загрузим модель из файла (структура + веса) и продолжим обучение

Ввод [8]:
```
del top_model
top_model = keras.models.load_model("mobilenet.clouds.h5")
```

Ввод [9]:
```
top_model.fit(base_model_output,
              load_y(train).reshape(len(train), -1), epochs=100,
              callbacks=[EarlyStopping(monitor="loss",
                 min_delta=0.0001, patience=5, verbose=1, mode="auto"),
                 ModelCheckpoint("mobilenet.clouds.h5", mode="auto",
                 monitor="val_loss", verbose=1)])
```

```
Epoch 00059: saving model to mobilenet.clouds.h5
Epoch 60/100
4436/4436 [==============================] - 142s 32ms/step - loss: 0.2602

Epoch 00060: saving model to mobilenet.clouds.h5
Epoch 61/100
4436/4436 [==============================] - 143s 32ms/step - loss: 0.2607

Epoch 00061: saving model to mobilenet.clouds.h5
Epoch 62/100
4436/4436 [==============================] - 141s 32ms/step - loss: 0.2604

Epoch 00062: saving model to mobilenet.clouds.h5
Epoch 63/100
4436/4436 [==============================] - 143s 32ms/step - loss: 0.2608

Epoch 00063: saving model to mobilenet.clouds.h5
Epoch 00063: early stopping
```

Out[9]:
```
<keras.callbacks.History at 0x36c7d288>
```

Соберем модель из обученного финального слоя и базовой модели

Ввод [10]:
```
model = Model(inputs=base_model.input,
              outputs=top_model(base_model.output))
model.compile(optimizer="adam", loss="mean_absolute_error")
model.summary()
```

```
block_16_depthwise_BN (None, 11, 17, 960)  3840       block_16_depthwise[0][0]
_________________________________________________________________
block_16_depthwise_relu (ReLU)  (None, 11, 17, 960)  0          block_16_depthwise_BN[0][0]
_________________________________________________________________
block_16_project (Conv2D)       (None, 11, 17, 320)  307200     block_16_depthwise_relu[0][0]
_________________________________________________________________
block_16_project_BN (BatchNorma (None, 11, 17, 320)  1280       block_16_project[0][0]
_________________________________________________________________
Conv_1 (Conv2D)                 (None, 11, 17, 1280) 409600     block_16_project_BN[0][0]
_________________________________________________________________
Conv_1_bn (BatchNormalization)  (None, 11, 17, 1280) 5120       Conv_1[0][0]
_________________________________________________________________
out_relu (ReLU)                 (None, 11, 17, 1280) 0          Conv_1_bn[0][0]
_________________________________________________________________
sequential_1 (Sequential)       (None, 25)           6941465    out_relu[0][0]
=================================================================================================
Total params: 9,199,449
Trainable params: 8,686,617
Non-trainable params: 512,832
_________________________________________________________________
```

Построение предсказания

Ввод [11]:
```
prediction = model.predict_generator(load_data(test, 1),
                                     steps=len(test), verbose=1)
```

```
1110/1110 [==============================] - 1462s 1s/step
```

Ввод [12]:
```
prediction = (prediction>0.5).astype("i1")
```

Ввод [13]:
```
print (prediction[0:10])
print (load_y(test.head(10)).reshape(10, -1))
```

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
[[0 1 1 1 1 0 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1]
 [1 1 1 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0]]
```

Ввод [20]:
```
target = []
masks = []
for i,vals in enumerate(prediction):
    if vals.sum() > 4:
        targ = 1
    else:
        targ = 0
    target.append(targ)
    masks.append(np.array2string(vals.flatten().astype("int8"),
                                 separator="")[1:-1])
test["MaskPixels"] = masks
test["target"] = target
print (test[test["target"]>0][["EncodedPixels","MaskPixels"]].head(20))
```

```
                                                  EncodedPixels  \
2216   1 1 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0
4656   0 0 1 0 0 0 1 1 1 0 0 1 1 1 0 1 1 0 0 0 0 0 1 0 0
4660   1 1 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0
5632   1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0
9096   0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1
7876   1 1 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0
7240   0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 1 0
552    0 0 1 0 1 0 1 1 1 0 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0
6800   1 1 1 0 0 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 0 0 1
20688  0 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 1 0 0 1 1 1 0 0
2208   0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 0
16520  0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0
7512   1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 1 1 1 0 1 0 1 1 0
308    0 1 1 1 1 0 1 1 1 1 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0
5376   0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 1 1
7616   1 1 1 0 0 1 1 1 0 1 1 0 1 0 0 1 1 0 1 0 1 0 1 0 1
5332   1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 0 0 0 1 1 0
15632  1 1 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0
980    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

                                                  MaskPixels
2216   1100011100111000111000000000
4656   0010001110011101100001000000
4660   1100011100111000111000000000
5632   1111110011101111111100000000
9096   0001000001000001100000101000
7876   1100011100111000111000000000
7240   0110001111111000111100010000
552    0010101110111001100111100000
6800   1110011100111101111010000100
20688  0000001100111001010011100000
2208   0000011100010001110000000000
16520  0100110000000001100110000000
7512   1000011000101101111010110000
308    0111101111011100111000000000
5376   0000000000000010011100111000
7616   1110011010110101010101010100
5332   1000011000101100111000110000
15632  1100011100111000111000000000
980    1111111111111111111111111111
```

Расчет точности предсказания

Нет облаков - 0.5, MLP - 0.3, CONV/VGG - 0.48, AlexNet - 0.21, Inception - 0.4, ResNet - 0.52, VGG16+Inception+ResNet - 0.54

Ввод [17]:
```
dice = test.apply(calc_dice, axis=1, result_type="expand")
print ("Keras, MobileNet:", round(dice.mean(), 3))
```

```
Keras, MobileNet: 0.472
```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Wolpert, David H. Stacked generalization // Neural networks 5.2 (1992): 241–259.
2. Breiman, Leo. Bagging predictors // Machine learning 24.2 (1996): 123–140.
3. Breiman, Leo. Random Forests // Machine Learning, 45(1), 5-32, 2001.
4. Freund, Yoav, Robert Schapire, and N. Abe. A Short Introduction to Boosting // Journal-Japanese Society For Artificial Intelligence 14.771–780 (1999): 1612.
5. Friedman, Jerome H. Stochastic gradient boosting // Computational Statistics and Data Analysis, 2002.
6. Журавлёв Ю. И. Об алгебраическом подходе к решению задач распознавания или классификации // Проблемы кибернетики. – 1978.
7. Skurichina M., Duin R. P. W. Limited bagging, boosting and the random subspace method for linear classifiers // Pattern Analysis & Applications. 2002. Pp. 121–135.
8. Jahrer, Michael. Netflix Prize report 2009. URL: http://elf-project.sourceforge.net/CombiningPredictionsForAccurateRecommenderSystems.pdf.
9. Toscher, Andreas and Jahrer, Michael. The BigChaos Solution to the Netflix Grand Prize // 2009. URL: http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf.
10. Menahem, Eitan, Lior Rokach, and Yuval Elovici. Troika-An improved stacking schema for classification tasks // Information Sciences 179.24 (2009): 4097–4122.
11. Seewald, A. How to Make Stacking Better and Faster While Also Taking Care of an Unknown Weakness // Nineteenth International Conference on Machine Learning. – P. 554–561. – 2002.
12. Allen Downey, Jeffrey Elkner, Chris Meyers. How to Think Like a Computer Scientist. – Green Tea Press, 2008. – 250 с.
13. Andreas C. Miller, Sarah Guido. Introduction to Machine Learning with Python. – O'Reilly Media, 2017. – 367 с.
14. Уэс Маккинни. Python и анализ данных. – O'Reilly Media, 2015. – 466 с.
15. Орельен Жерон. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. – O'Reilly Media, 2018. – 662 с.
16. Дж. Клейнберг, Е. Тардос. Алгоритмы: Разработка и применение. – СПб.: Питер, 2016 – 800 с.
17. Педро Домингос. Верховный алгоритм. – Манн, Иванов и Фербер. 2016 – 315 с.
18. Wiley Brand. Python for Data Science for Dummies. – John Willey & Sonc, 2015. – 407 с.
19. The Python Manual – Black Dog i-Tech Series, 2018. – 160 с.
20. Brett Slatkin. Effective Python – Addison Wesley, 2015. – 650 с.
21. Yeradis P. Barbosa Marrero. The Python Tutorial 2.7. – Leanpub, 2014. – 127 с.
22. Arun Tigeraniya. Python Unlocked. – Packt Publishing, 2015. – 147 с.
23. Линейная регрессия. Википедия – открытая энциклопедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Линейная_регрессия.

24. Основы линейной регрессии [Электронный ресурс]. URL: http://statistica.ru/theory/osnovy-lineynoy-regressii/.
25. Полное руководство по линейной регрессии в Scikit Learn [Электронный ресурс]. URL: https://pythonru.com/uroki/linear-regression-sklearn.
26. Библиотека Pandas – типы данных [Электронный ресурс]. URL: https://mipt-stats.gitlab.io/courses/python/08_pandas1.html#5.-Интервалы-времени.
27. Sklearn.linear_model. LinearRegression [Электронный ресурс]. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html.
28. Изучение LVQ [Электронный ресурс]. Режим доступа: https://coderlessons.com/tutorials/akademicheskii/izuchite-iskusstvennuiu-neironnuiu-set/izuchenie-vektora-kvantovaniia.
29. Fabrizio Romano. Learning Python. – Packt Publishing, 2015. – 405 с.
30. Майкл Солтис. Введение в анализ алгоритмов. – СПб.: ДМК, 2019. – 269 с.
31. David Julian. Designing Machine Learning Systems with Python. – Packt Publishing, 2016. – 209 с.
32. Rick van Hattem. Mastering Python. – Packt publishing, 2016. – 451 с.
33. Obi Ike-Nwosu. Intermediate Python. – Leanpub, 2016. – 169 с.
34. B. J. Korites. Python Graphics. – Apress, 2018. – 359 с.
35. Дэвид Колец. Классические задачи Computer Science на языке Python. – СПб.: Питер, 2020. – 256 с.
36. Danish Haroon. Python Machine Learning Case Studies. – Apress, 2017. – 201 с.
37. Н. А. Прохоренок, В. А. Дронов. Python 3: Самое необходимое. – СПб.: БХВ-Петербург, 2019. – 608 с.
38. Mark Lutz. Python Pocket Reference. – O'Reilly Media, 2014. – 243 с.
39. Мэтт Харрисон. Как устроен Python. – СПб.: Питер, 2019. – 272 с.
40. Python Уроки Второе издание.
41. Richard Moore. Python Machine Learning. – O'Reilly Media, 2019. – 254 с.
42. Dmitry Zinovier. Complex Network Analysis in Python. – The Pragmatic Bookshelf, 2018. – 219 с.
43. Марк Лутц. Изучаем Python. – СПб.: ООО «Диалектика», 2019. – 832 с.
44. Томас Кормен, Чарльз Лейзерсон. Алгоритмы: построение и анализ, 3-е издание. – М.: ООО И.Д. Вильямс, 2013. – 1328 с.
45. Dimitrios Kouzis-Loukas. Learning Scrapy. – Packt Publishing, 2016. – 243 с.
46. Ной Гифт. Программный ИИ. – СПб.: Питер, 2019. – 304 с.
47. Доля П. Г. Введение в научный Python. – Харьковский Национальный Университет, 2016. – 265 с.
48. Sergio Rojas. Prealgebra VIA Python Programming. – Caracas, 2018. – 257 с.
49. Andrew N. Harrington. Hands-on Python Tutorial. – Loyola eCommons, 2015. – 170 с.
50. Daniel Arbuckle. Learning Python Testing. – Packt Publishing, 2014. – 314 с.
51. Джон Пол Мюллер, Лука Массарон. Искусственный интеллект для чайников. – СПб.: Диалектика, 2017. – 354 с.
52. Samir Madhavan. Mastering Python for Data Science. – Packt Publishing, 2018. – 276 с.
53. Joe Thompson. Python's Companion. – Packt Publishing, 2015. – 211 с.

54. Mark J. Johnson. A Concise Introduction to Programming in Python Second Edition. – CRC Press, 2018. – 197 с.
55. Освальдо Мартин. Байесовский анализ на Python. – М.: ДМК Пресс, 2020. – 340 с.
56. Coding for Python. – Black Dog i-Tech Series, 2019. – 160 с.
57. Sunil Kapil. Clean Python. – Apress, 2019. – 261 с.
58. Джульен Данжу. Путь Python. – СПб.: Питер, 2020. – 256 с.
59. Wiley Brand. Python all-in-one. – Dummies, 2019 – 647 с.
60. Кэмерон Дэвидсон-Пайлон. Вероятное программирование на Python. Байесовский вывод и алгоритмы. – СПб.: Питер, 2019. – 256 с.
61. David Kopec. Classic Computer Science Problems in Python. – Manning Shelter Island, 2019. – 201 с.
62. Steven L. Gordon and Brian Guilfoos. Introduction to Modeling and simulation with MATLAB and Python. – CRC Press, 2017. – 183 с.
63. Дэвид Бизли, Брайан К. Джонс. Python Книга рецептов. – М.: ДМК Пресс, 2019. – 648 с.
64. Naomi Ceder. The Quick Python Book. – Manning Press, 2018. – 431 с.
65. Bastiaan Sjardin, Luca Massaron. Large Scale Machine with Python. – Packt publishing, 2018. – 358 с.
66. Robert Johansson. Numerical Python. – Apress, 2018. – 358 с.
67. Moshe Zadza. DevOps in Python. – Apress, 2019 – 165 с.
68. Yinyan Zhang. Deep Reinforcement Learning with Guaranteed Performance. – Springer Press, 2020. – 265 с.
69. Брайан Макмахан, Делип Рао. Знакомство с PyTorch. – СПб.: Питер, 2020 – 256 с.
70. Amit Nandi. Spark for Python Developers. – Packt Publishing, 2018. – 219 с.
71. Ivan Idris. Python Data Analysis. – Packt Publishing, 2014. – 319 с.
72. Serge Kruk. Practical Python Al Projects. – Apress, 2018. – 271 с.
73. Fabio Nelli. Python Data Analytics. – Apress, 2015. – 331 с.
74. Richard L. Halterman. Fundamentals of Python Programming. – Packt Publishing, 2014. – 269 с.
75. Luke Sneeringer. Professional Python. – Manning Press, 2016. – 321 с.
76. Gabriele Lanaro. Python High Performance Programming. – Packt Publishing, 2013. – 93 с.
77. Framcisco J. Blanco-Silva. Mastering SciPy. – Packt Publishing, 2015. – 375 с.
78. Donald J. Norris. Beginning Artificial Intelligence with the Raspberry Pi. – Apress, 2017. – 363 с.
79. Sam Abrahams, Danijar Hafner. Tensorflow For Machine Intelligence. – Bleeding Edge Press, 2016. – 245 с.
80. Pratap Dangeti, Allen Yu, Claire Chung. Learning Path Numerical Computing with Python. – Packt Publishing, 2018. – 419 с.
81. J. T. Wolohan. Mastering Large Datasets with Python. – Mannig Press, 2019. – 365 с.
82. Yves Hilpisch. Derivatives Analytics with Python. – Willey Press, 2015. – 347 с.
83. Cay Horstmann, Rance Necaise. Python for Everyone. – Willey Press, 2012. – 311 с.

84. Massimiliano Pippi. Python for Google App Engine. – Packt Publishing, 2015. – 175 с.
85. Learning Vector Quantization [Электронный ресурс]. URL: https://www.geeksforgeeks.org/learning-vector-quantization/.
86. Реализация Learning Vector Quantization [Электронный ресурс]. URL: https://machinelearningmastery.com/implement-learning-vector-quantization-scratch-python/.
87. Learning Vector Quantization [Электронный ресурс]. URL: https://towardsdatascience.com/learning-vector-quantization-ed825f8c807d.

ВАРИАНТЫ ЗАДАНИЙ ДЛЯ САМОСТОЯТЕЛЬНОЙ РЕАЛИЗАЦИИ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ

В данном блоке задач представлены ссылки на исходные наборы данных. При использовании этих отобранных данных вы сможете решить поставленную задачу (сформулированную в описании каждой URL ссылки).

Задачи, представленные в Приложении 1, отобраны таким образом, что они не требуют от вас дополнительных усилий по аудиту исходного датасета, включения в него непрофильных сведений и прочего. Датасет является единственным источником данных, что существенно упрощает задачу, позволяя сконцентрировать всё своё внимание на реализации алгоритма машинного обучения (алгоритм определяется по полученному от преподавателя варианту).

1. Задача из медицинской сферы по определению бокового амиотрофического склероза.
   URL: https://www.kaggle.com/alsgroup/end-als?select=end-als

2. Задача детекции цветка одуванчика и обособление его от других видов цветов (растений)
   URL: https://www.kaggle.com/coloradokb/dandelionimages

3. Определение наличия сердечного заболевания
   URL: https://www.kaggle.com/ronitf/heart-disease-uci

4. Набор данных по определению элементов одежды
   URL: https://www.kaggle.com/zalando-research/fashionmnist

5. Анализ текстов отзывов покупателей от крупной коммерческой торговой площадки
   URL: https://www.kaggle.com/olistbr/brazilian-ecommerce

6. Задача предсказания дождя на следующий день в Австралии
   URL: https://www.kaggle.com/jsphyg/weather-dataset-rattle-package

7. Набор данных, содержащий более 90 000 изображений фруктов. Задача определения его названия
   URL: https://www.kaggle.com/moltean/fruits

8. Задача определения вида (класса) птицы. Набор содержит 315 видов птиц.
   URL: https://www.kaggle.com/gpiosenka/100-bird-species

9. Мультиклассовый тест на классификацию одного изображения
   URL: https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

10. Определение знаменитостей по фотографии лица
    URL: https://www.kaggle.com/jessicali9530/celeba-dataset

11. Задача отдела кадров в машинном обучении: спрогнозировать вероятность того, что кандидат будет искать новую работу или останется работать в компании
    URL: https://www.kaggle.com/arashnic/hr-analytics-job-change-of-data-scientists

12. Определение дермоскопических поражений кожи
    URL: https://www.kaggle.com/kmader/skin-cancer-mnist-ham10000

13. Набор данных реальных и поддельных объявлений о вакансиях на трудоустройство
    URL: https://www.kaggle.com/shivamb/real-or-fake-fake-jobposting-prediction

14. Определение аритмии или инфаркта по данным сердцебиения
    URL: https://www.kaggle.com/shayanfazeli/heartbeat

15. Обучение модели на заголовках новостей
    URL: https://www.kaggle.com/rmisra/news-category-dataset

16. Определение стоимости жилья в пригородах Бостона
    URL: https://www.kaggle.com/arslanali4343/real-estate-dataset

17. Диагностика заболеваний крови по ее образцам
    URL: https://www.kaggle.com/paultimothymooney/blood-cells

18. Набор данных заголовков новостей для обнаружения в них сарказма
    URL: https://www.kaggle.com/rmisra/news-headlines-dataset-for-sarcasm-detection

19. Набор данных по определению Стэнфордских собак
    URL: https://www.kaggle.com/jessicali9530/stanford-dogs-dataset

20. Задача определения типа еды (от яблока до вафель)
    URL: https://www.kaggle.com/kmader/food41

21. Изображения всех покемонов от поколения 1 до поколения 7, а также их типы (первичный и вторичный) в виде csv
    URL: https://www.kaggle.com/vishalsubbiah/pokemon-images-and-types

22. Крупномасштабный набор данных для сегментации и классификации рыб
    URL: https://www.kaggle.com/crowww/a-large-scale-fish-dataset

23. Набор данных музыки. В задачи входит определения ноты / инструмента / композитора
    URL: https://www.kaggle.com/imsparsh/musicnet-dataset

24. Набор автомобилей Стэнфорда, в задачи входит определение марки
    URL: https://www.kaggle.com/jessicali9530/stanford-cars-dataset

25. Естественные изображения. Скомпилированный набор данных из 6899 изображений из 8 различных классов.
    URL: https://www.kaggle.com/prasunroy/natural-images

ВАРИАНТЫ ЗАДАНИЙ ДЛЯ ИССЛЕДОВАТЕЛЬСКИХ РАБОТ В ОБЛАСТИ МАШИННОГО ОБУЧЕНИЯ

В данном наборе:

1. Обнаружение и классификация индийских номерных знаков.
   URL: https://www.kaggle.com/dataclusterlabs/indian-number-plates-dataset

2. Каскады Хаара для распознавания лиц.
   URL: https://www.kaggle.com/gpreda/haar-cascades-for-face-detection

3. Anime Faces.
   URL: https://www.kaggle.com/soumikrakshit/anime-faces

4. Гистопатологические изображения рака легких и толстой кишки; также «здоровые» снимки.
   URL: https://www.kaggle.com/andrewmvd/lung-and-colon-cancer-histopathological-images

5. Мульти-классный набор данных для обучения компьютерному зрению.
   URL: https://www.kaggle.com/sanikamal/rock-paper-scissors-dataset

6. Обнаружение рекламы на YouTube.
   URL: https://www.kaggle.com/mathurinache/youtube-ads-detection

7. WLASL (американский язык жестов мирового уровня). 12 000 обработанных видеороликов с глоссарием американского жестового языка на уровне слов.
   URL: https://www.kaggle.com/risangbaskoro/wlasl-processed

8. Набор данных об опухолях Брайана.
   URL: https://www.kaggle.com/preetviradiya/brian-tumor-dataset

9. Набор данных кошек. Более 9000 изображений кошек с аннотированными чертами лица.
   URL: https://www.kaggle.com/crawford/cat-dataset

10. Набор данных изображения для алфавитов американского жестового языка.
    URL: https://www.kaggle.com/grassknoted/asl-alphabet

ВАРИАНТЫ ЗАДАНИЙ, ВКЛЮЧАЮЩИЕ В СЕБЯ САМОСТОЯТЕЛЬНЫЙ ЭТАП DATA MINING, ДЛЯ ПОСТРОЕНИЯ END-TO-END РЕШЕНИЙ В ОБЛАСТИ МАШИННОГО ОБУЧЕНИЯ

Данная глава подразумевает отличный уровень владения инструментами и навыками машинного обучения. Для выполнения задания высока вероятность того, что вам потребуются дополнительные знания, кроме тех, что уже существуют в наборе данных.

Где найти эти данные, каким образом их сконфигурировать и стоит ли расширять набор – этот выбор остается за вами. Результат точности модели должен быть подтвержден метриками качества и удовлетворять условиям внедрения в прикладную область.

Ориентировочная точность модели машинного обучения разнится от задачи к задаче, но примерно должна быть не менее 90–99 %.

1. Определение возраста, пола и этноса по фотографии лица.
   URL: https://www.kaggle.com/nipunarora8/age-gender-and-ethnicity-face-data-csv

2. Изображения CAPTCHA. Вам необходимо создать модель, которая будет проходить проверку в автоматическом режиме.
   URL: https://www.kaggle.com/fournierp/captcha-version-2-images

3. Набор данных из базы данных биометрических отпечатков пальцев, разработанный для академических исследований. Необходимо определить синтетически измененные отпечатки пальцев.
   URL: https://www.kaggle.com/ruizgara/socofing

4. Обнаружение бактерий с помощью микроскопии темного поля. Набор данных для сегментации спирохет с изображениями и вручную аннотированными масками.
   URL: https://www.kaggle.com/longnguyen2306/bacteria-detection-with-darkfield-microscopy

5. Автоматическое трансформирование черно-белого фото в цветное. Набор данных черно-белых изображений.
   URL: https://www.kaggle.com/shravankumar9892/image-colorization

6. Определение местоположения по аэрофотоснимку на основании участков снимков штата Массачусетс.
   URL: https://www.kaggle.com/balraj98/massachusetts-roads-dataset

7. Набор данных для распознавания арабских рукописных символов.
   URL: https://www.kaggle.com/mloey1/ahdd1

8. Сегментирование всего тела с изображения (удаление любого фона с любой фотографии в автоматическом режиме).
   URL: https://www.kaggle.com/tapakah68/segmentation-full-body-mads-dataset

9. Задача создания системы для автоматического определения штампов (печатей) и их верификация при разных условиях (поворот, цвет сканирования, качество изображения).
   URL: https://www.kaggle.com/rtatman/stamp-verification-staver-dataset

10. Локализация сканирования кошек.
    URL: https://www.kaggle.com/uciml/ct-slice-localization

11. Определение маски на лице человека.
    URL: https://www.kaggle.com/andrewmvd/face-mask-detection

12. Набор данных содержит векторные изображения некоторых популярных логотипов брендов.
    URL: https://www.kaggle.com/kkhandekar/popular-brand-logos-image-dataset

Учебное издание

Андрей Владимирович Протодьяконов
Пётр Андреевич Пылов
Владимир Евгеньевич Садовников

АЛГОРИТМЫ DATA SCIENCE
И ИХ ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ НА PYTHON

Учебное пособие

ISBN 978-5-9729-1006-9

Подписано в печать 25.02.2022
Формат 60×84/16. Бумага офсетная.
Гарнитура «Times New Roman».

Издательство «Инфра-Инженерия»
160011, г. Вологда, ул. Козленская, д. 63
Тел.: 8 (800) 250-66-01
E-mail: booking@infra-e.ru
https://infra-e.ru