Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности

_____ С.Т. Князев

«27» декабря 2021 г.

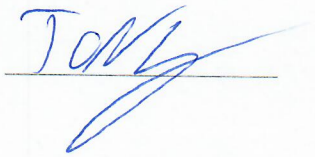
Машинное обучение

Учебно-методические материалы по направлению подготовки
09.04.01 Информатика и вычислительная техника
Образовательная программа «Инженерия искусственного интеллекта»

Екатеринбург

2021

РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Доцент, канд.техн.наук  Долганов Антон Юрьевич

СОДЕРЖАНИЕ

1. **История и Базовые Понятия**

Список тем, которые должны быть осуждены на лекции:

1. Что такое машинное обучение
2. История машинного обучения
3. Основные термины в машинном обучении
4. Классификация задач машинном обучении

Ключевые моменты по темам:

1. Обсуждение того, где можно встретить результаты машинного обучения (задать студентам вопросы вида «когда в последний раз вы сталкивались с результатами машинного обучения»)

   Обсуждение определений «машинное обучение», «искусственный интеллект», «интеллект»

   Сравнение подхода, основанного на традиционном программировании с машинным обучением
2. Историческая ретроспектива об искусственном интеллекте и машинном обучении. Обсуждение «взлетов и падений» подходов, основанных на машинном обучении.

   Текущее состояние искусственного интеллекта и машинного обучения
3. Обсуждение основных терминов, связанных с машинным обучением
   - Объект, Цель, Признаки
   - Тренировочная и Тестовая выборка
   - Модель, параметры и гиперпараметры модели
   - Функция потерь и цель обучения
   - Разложение ошибок модели на смещение и дисперсию. Переобучение
   - Валидация моделей. Отложенная выборка и кросс-валидация
4. Обсуждение типовых задач машинного обучения и ключевых различий между ними.

   Обучение с учителем: задачи регрессии и классификации

Обучение без учителя: задачи кластеризации и уменьшения размерности

Обучение с подкреплением

Обсуждение примеров различных задач машинного обучения

Подведение Итогов Лекции

Текстовый материал

В Национальной стратегией развития искусственного интеллекта на период до 2030 года, утвержденной Указом Президента Российской Федерации от 10 октября 2019 г. № 490 «О развитии искусственного интеллекта в Российской Федерации» можно увидеть следующее определение для искусственного интеллекта:

«**Искусственный интеллект**» – комплекс технологических решений, позволяющий имитировать когнитивные функции человека (включая самообучение и поиск решений без заранее заданного алгоритма) и получать при выполнении конкретных задач результаты, сопоставимые, как минимум, с результатами интеллектуальной деятельности человека.

При этом комплекс технологических решений включает в себя информационно-коммуникационную инфраструктуру, программное обеспечение (в том числе, в котором используются методы машинного обучения), процессы и сервисы по обработке данных и поиску решений

Tom M. Mitchel “A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.”

Машинное обучение рассматривается в простой концепции $y = f(X)$, где X – исходные данные; f – некое преобразование, которое позволяет получить y – ожидаемый ответ.

Ключевые понятия стоит обсудить с конкретного примера данных. На Рис. 1-1 представлены данные для некоторой совокупности студентов.

id Студента	Пол	Возраст	Институт	Общежитие	Работа	Оценка Python	ЕГЭ Инф.	Быллы по МО	Экзамен по МО
0	Ж	24	ФТИ	нет	нет	75	83	54	Отл.
1	М	23	Другой	нет	нет	79	40	98	Уд.
2	М	24	Другой	нет	да	43	59	43	Уд.
3	Ж	24	ИРИТ-РТФ	нет	да	98	83	46	Отл.
4	М	24	ИРИТ-РТФ	да	да	50	65	49	Неуд.
5	М	24	ФТИ	да	да	45	96	90	Неуд.
6	Ж	23	ИРИТ-РТФ	нет	нет	71	98	50	Хор.
7	Ж	23	ИРИТ-РТФ	нет	нет	98	43	55	Уд.
8	Ж	24	ИРИТ-РТФ	да	да	49	61	83	Неуд.
9	Ж	24	ИЕНиМ	да	да	63	46	71	Хор.

П

строки);

- y цель (target), которая является ожидаемым предсказанием (на Рис. 1-1– две целевые переменные для конкретного студента, баллы по курсу Машинное обучение и оценка за экзамен по машинному обучению);
- Y полный набор целей (на Рис. 1-1– целевые переменные для совокупности студентов, оба столбца);
- *Признаки* (англ. *features*) – что-то, что описывает объекты (на Рис. 1-1– это пол, возраст, институт, общежитие, работа, Баллы ЕГЭ по информатике и Оценка за курс по Python).

В связи с этим, на высоком уровне типичная задача машинного обучения формулируется следующим образом: искать возможности получать целевые переменные при использовании признакового пространства данных.

Обучение модели

В предыдущем разделе мы часто употребляли понятие модель. В общем случае **модель** использует некие операции над признаками для предсказания целевой переменной, т. е. происходит отображение из пространства признаков в пространство целевых предсказаний:

$a: X \rightarrow Y$,

где $a \in A$ семейство Моделей.

Обычно предсказания модели обозначают символом $\hat{y}$ (игрек с «крышечкой»). В общем случае справедливо следующее выражение

$$\hat{y}_i = a\left(x_i, w, h\right),$$

где x_i – признаки для i-ого объекта, w параметры Модели (оптимизируются алгоритмом модели), h гиперпараметры модели (оптимизируются нами – Data Scientist'ами, экспертами по машинному обучению). Более подробно о параметрах и гиперпараметрах моделей мы обсудим в последующих главах.

После того, как мы выбрали модель ее необходимо обучить. Следующая концепция, которую стоит обсудить это **Тренировочная** и **Тестовая выборка**. **Тренировочная выборка** — это такой набор данных, для которого нам известны пары «признаки» - «целевая переменная» для каждого субъекта из выборки. **Тестовая выборка** – это такой набор данных, для которого известны только признаки. Например (Рис. 1-2), у нас есть данные по студентам набора 2021 года, по которым мы знаем всю статистику. И к нам в 2022 году поступили новые студенты. И мы хотим, используя опыт предыдущего года попытаться предсказать, студентов у которых могут быть проблемы с курсом машинное обучение, и наоборот, студенты, которые успешно справятся и им лучше выдавать задачки посложнее.

Тренировочная выборка

id Студента	Пол	Возраст	Институт	Общежитие	Работа	Оценка Python	ЕГЭ Инф.	Быллы по МО	Экзамен по МО
0	Ж	24	ФТИ	нет	нет	75	83	54	Отл.
1	М	23	Другой	нет	нет	79	40	98	Уд.
2	М	24	Другой	нет	да	43	59	43	Уд.
3	Ж	24	ИРИТ-РТФ	нет	да	98	83	46	Отл.
4	М	24	ИРИТ-РТФ	да	да	50	65	49	Неуд.
5	М	24	ФТИ	да	да	45	96	90	Неуд.
6	Ж	23	ИРИТ-РТФ	нет	нет	71	98	50	Хор.
7	Ж	23	ИРИТ-РТФ	нет	нет	98	43	55	Уд.
8	Ж	24	ИРИТ-РТФ	да	да	49	61	83	Неуд.
9	Ж	24	ИЕНиМ	да	да	63	46	71	Хор.

Тестовая выборка

id Студента	Пол	Возраст	Институт	Общежитие	Работа	Оценка Python	ЕГЭ Инф.	Быллы по МО	Экзамен по МО
10	М	23	ИЕНиМ	да	да	51	84	?	?
11	Ж	24	ИЕНиМ	да	нет	90	44	?	?
12	Ж	24	ФТИ	да	нет	60	72	?	?
13	Ж	24	ИЕНиМ	да	нет	55	76	?	?
14	Ж	23	ФТИ	нет	да	40	54	?	?
15	М	24	Другой	нет	да	94	64	?	?
16	М	23	Другой	да	да	65	46	?	?
17	М	24	Другой	нет	нет	50	49	?	?
18	Ж	23	ИРИТ-РТФ	да	да	62	93	?	?
19	М	23	ИЕНиМ	да	да	56	40	?	?

Рис. 1-2 Тренировочная и тестовая выборка

Для того чтобы оценить, насколько плоха или хороша конкретная модель нам необходимо воспользоваться **функцией потерь** (Loss) $L(y_i, \hat{y}_i)$, для оценки качества модели. В зависимости от типа целевой переменной (числовая или категориальная) функции потери могут отличаться, но в них есть общая идеология. Если для конкретного объекта $y_i \sim \hat{y}_i$, т. е. предсказание модели и реальное значения совпадают, тогда функция потерь $L\left(y_i, \hat{y}_i\right)$ принимает небольшие значения, иначе, если предсказание модели разнятся, то функция потерь принимает большие значения.

Используя данные Тренировочной выборки, мы можем оценить **Функционал Потерь**, среднее значения функции потерь для всех объектов из тренировочной выборки $Q(a, X) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, a\left(x_i, w, h\right)\right)$.

Таким образом можно сформулировать цель обучения следующим образом: $Q(a, X) \rightarrow$. Другими словами, в ходе обучения мы должны подобрать такие параметры и гиперпараметры модели, которые наилучшим образом предсказывают целевые значения в тренировочной выборке. В общем случае

простые модели (которые содержат ограниченное число признаков, простые зависимости между переменными) обладают большими значениями функционала потерь, в то же время сложные модели (в которых много признаков и существуют сложные зависимости между переменными) могут иметь сколь угодно низкие значения функционала потерь. В общем случае зависимость функционала потерь от сложности модели представлена на Рис. 1-3.

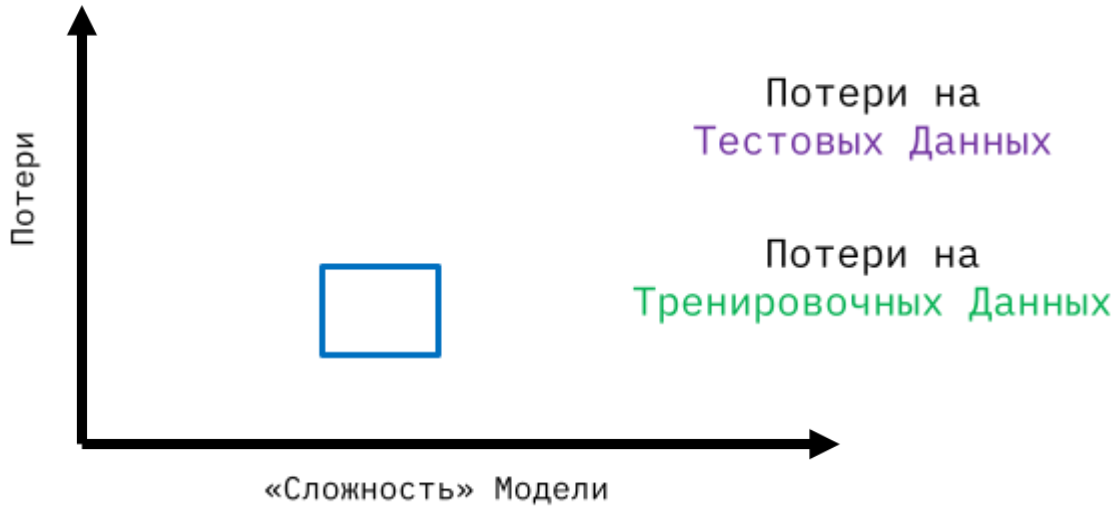


Рис. 1-3 Зависимость потерь от сложности моделей

Здесь стоит отметить, что практической пользы, от модели, которая хорошо запоминает тренировочную выборку. Поэтому обычно потери на тренировочных данных рассматривают совместно с потерями на тестовых данных. При усложнении модели потери на тестовых данных как правило тоже в начале убывают. Но в определенный момент может возникнуть ситуация, когда потери на тестовой выборке начинают снова расти, а потери на тренировочной выборке продолжают падать. Это явление называется **переобучением**. Чтобы избежать переобучения мы должны использовать тренировочную выборку для своевременной остановки алгоритма обучения и выбора оптимальных параметров и гиперпараметров модели. Но есть проблема: по умолчанию у нас нет значений целевых переменных для тестовой выборки.

Однако мы можем смоделировать тестовую выборку, используя подход, который называется **валидация**. Мы можем взять «кусочек» тренировочной выборки, отложить его, обучить модель на остальной тренировочной выборке и проверить на отложенном «кусочке». Такой подход называется использование **отложенной выборки** (Hold-Out split) Рис. 1-4.

Рис. 1-4 Схема отложенной выборки

Для повышения уверенности в модели можно повторить процесс разбиения на тренировочную и валидационную несколько раз, каждый раз разбивая данные несколько различным образом.

Или же можно воспользоваться подходом n-Fold Кросс-валидация (n-Fold Cross-Validation split) (Рис. 1-5). Обучающая выборка разбивается на n одинаковых по объему частей, которые содержат разные объекты. Производится n итераций. На каждой итерации происходит следующее:

- модель обучается на n−1 частях обучающей выборки;
- модель тестируется на части обучающей выборки, которая не участвовала в обучении.

Итоговая оценка функционала потерь усредняется по всем n итерациям. Как правило, n=10 (5 в случае малого размера выборки).

Рис. 1-5 Схема n-fold кросс-валидации

Так или иначе это позволяет смоделировать ситуацию, когда в модели подставляются новые, не виденные ранее данные. Поэтому потери на валидационных данных можно использовать для определения оптимальных параметров и гиперпараметров модели.

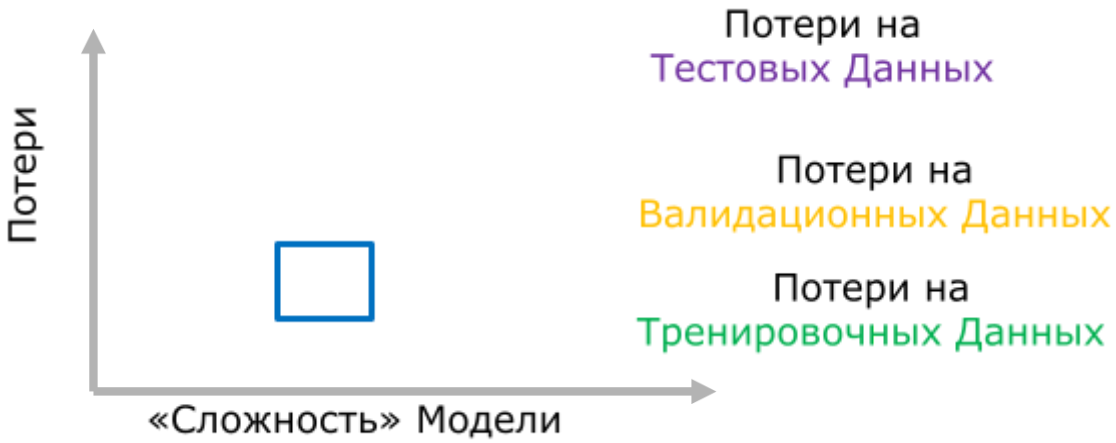


Рис. 1-6 Зависимость потерь от сложности моделей с учетом валидационных данных

Разложение Ошибки на Смещение и Дисперсию

В общем случае функционал потерь можно разложить на следующие составляющие:

$$Loss(a, X) \sim Bias(a(X)) + Variance(a(X)) + \sigma^2,$$

где σ^2- случайный шум, с которым мы, к сожалению, ничего не сможем сделать.

В связи с этим мы должны быть готовы, что модели не могут быть совершенными. Однако мы можем уменьшать другие слагаемы в данном разложении.

Ошибка **смещения** (**Bias** Error) — это ошибка из-за ошибочных предположений в алгоритме обучения. Высокая ошибка смещения - модель слишком проста. Ошибку смещения мы можем оценить, используя тренировочные данные.

Дисперсия (Variance) — это ошибка из-за чувствительности к небольшим колебаниям обучающей выборки. Высокая дисперсия - модель плохо работает на новых данных. Дисперсию модели мы можем оценить с использованием валидационных данных.

В зависимости от величины смещения и дисперсии существуют различные ситуации, которые можно описать с помощью мишеней, как указано на Рис. 1-7. Здесь близость к «яблочку» показывает наименьшее значения функции потерь для конкретного объекта.

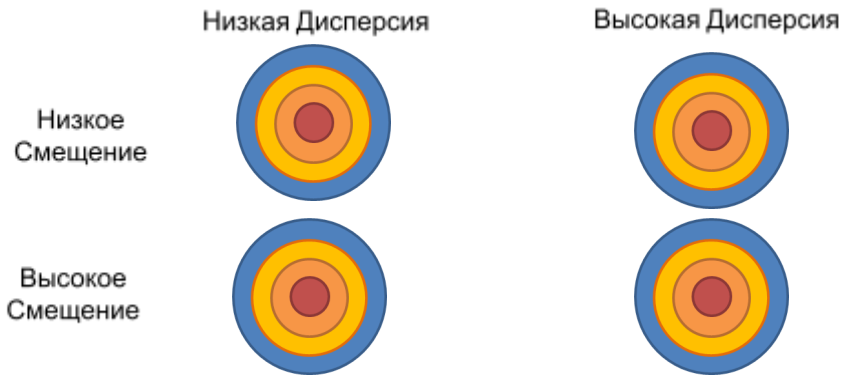


Рис. 1-7 Схематичное представление разложение ошибок модели

Идеальная ситуация – низкое смещение и низкая дисперсия. Модель хорошо работает в среднем и при этом для разных данных эта тенденция сохраняется. В случае высокого смещения и низкой дисперсии модель выдает стабильные предсказания как на тренировочной, так и на валидационной выборке, но, к сожалению, эти предсказания далеки от реальных значений. С другой стороны, ситуация низкого смещения и высокой дисперсии показывает то, что модель в среднем работает неплохо, однако существует достаточно большое количество отдельных объектов, на которых предсказания просто ужасны. Наконец худшая из возможных ситуаций – высокое смещение и высокая дисперсия, говорит о том, что модель совсем не подходит для имеющихся у нас данных.

Идеальные ситуации встречаются редко и поэтому необходимо находить компромисс между высоким смещением и высокой дисперсией. А это уже зависит от конкретной постановки задачи.

Задачи Машинного обучения

Настало время поговорить о типовых задачах Машинного обучения. Стандартно задачи машинного обучения разделяют на следующие:

- Обучение с учителем (Supervised learning)
- Обучение без учителя (Unsupervised learning)
- Обучение с подкреплением (Reinforcement learning)

Обучение с учителем: есть **обучающий** набор данных (тренировочная выборка). Для каждого **экземпляра** из набора данных есть пары входные данные/признаки и ожидаемый ответ. В этом случае задачей является поиск модели или "алгоритма", который предсказывает ожидаемые целевые ответы.

Далее возникают различные ситуации в зависимости от того, что мы ожидаем в качестве ожидаемых ответов. Если множество возможных ответов конечно, то речь идет о задаче **Классификации**:

- $Y \in \{1, ..., K\}$, $K \in Z$ в случае многих классов;
- $Y \in \{-1, +1\}$ или $Y \in \{0, 1\}$ в случае двух классов.

Схематично это представляется следующим образом: у нас есть некоторые пространство признаков, при этом существует заранее известно разметка (раскраска) отдельных объектов. И задача классификации сводится к построению такой разделяющей кривой, которая способна предсказывать метку или класс объекта Рис. 1-8.

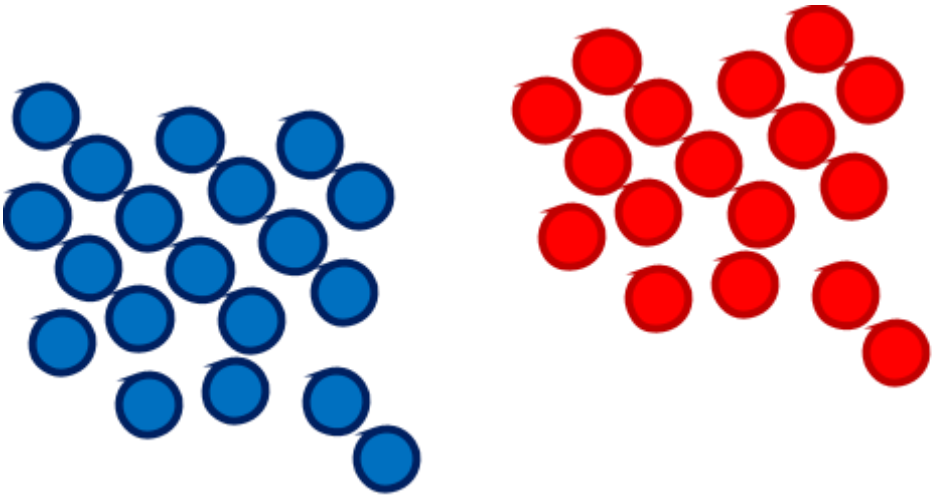


Рис. 1-8 Схематичное представление задачи классификации

На Рис. 1-1 Оценка за экзамен по Машинному обучению является целевой переменной для задачи классификации, т. к. количество возможных ответов конечно.

С другой стороны, множество возможных ответов может быть бесконечным, т. е. $Y \in R$. В таком случае решается задача **Регрессии**. Простой пример – у нас есть статистика по стоимости квартиры и ее площади. Но не для всех значений площадей. Используя имеющиеся данные, мы хотим предсказать стоимость квартиры для тех значений площадей, которых нет в нашей тренировочной выборке.

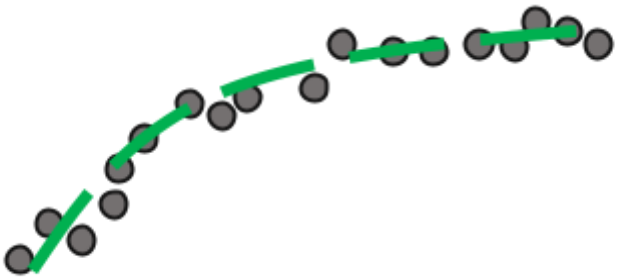


Рис. 1-9 Схематично представление задачи регресии

На Рис. 1-1 Балл за текущую успеваемость по Машинному обучению может быть целевой переменной для задачи регрессии, т. к. количество возможных ответов конечно.

Обучение без учителя: постановка задачи может смутить, так как изначально у нас есть только **данные.** При этом мы хотим что-то понять и найти закономерности в этих данных.

Например, задача **Кластеризации**. Мы хотим разметить принадлежность отдельных объектов к различным кластерам, используя, например, близость отдельных точек.

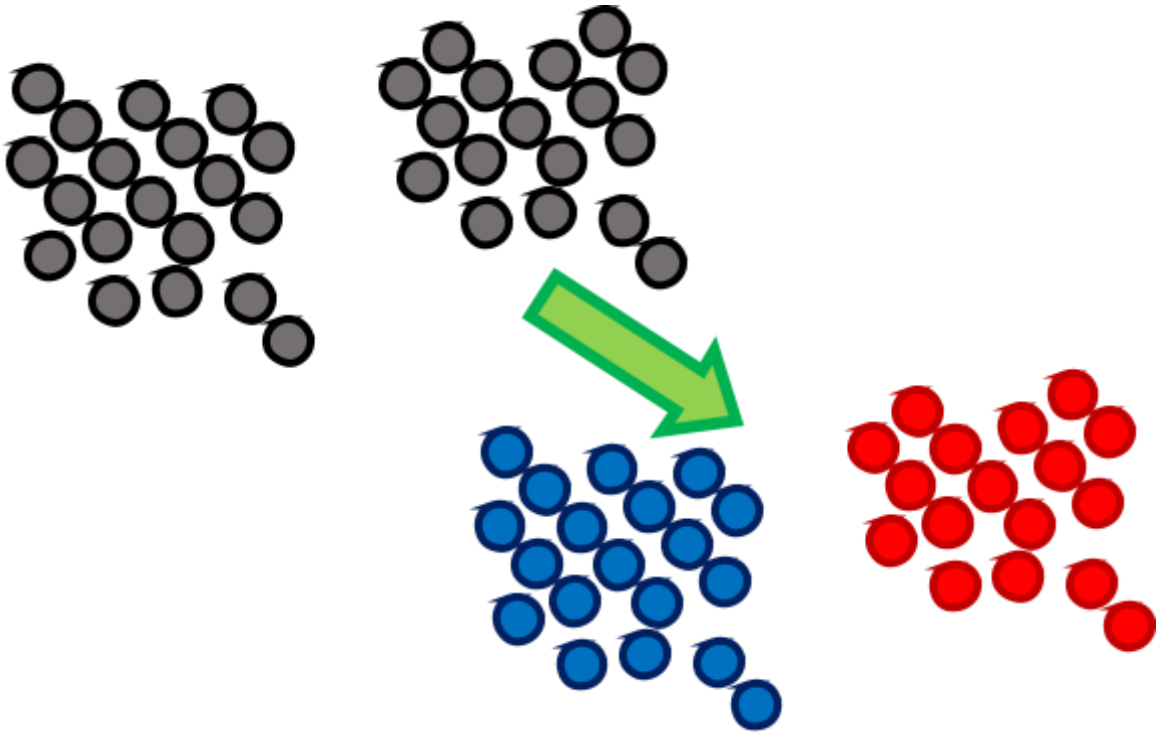


Рис. 1-10 Схематично представление задачи кластеризации

Стоит отметить, что в отличии от задачи классификации нам изначально не известны реальные классы объектов, и алгоритм должен принимать решение исключительно исходя из внутренних закономерностей в данных.

Другой типовой задачей обучения без учителя является задача **Снижения размерности**. Имеются табличные данные, большой размерности. И мы хотим построить такую новую таблицу данных, используя исходные данные, чтобы было проще с этим работать. Например, мы хотим сократить размерность табличных данных до 2 или 3, чтобы мы могли визуализировать имеющиеся у нас данные.

Обучение с подкреплением: есть среда и есть некоторая система, которая взаимодействует со средой. Задача состоит в эффективном взаимодействии со средой.

Ссылки на справочные материалы

1. Блог с описанием ключевых терминов и понятий машинного обучения «простыми словами» https://vas3k.ru/blog/machine_learning/
2. Хронология ключевых событий связанных с машинным обучением https://en.wikipedia.org/wiki/Timeline_of_machine_learning
3. Блог в котором освещаются некоторые важные прикладные аспекты машинного обучения, с пониманием которых у практиков часто возникают трудности https://dyakonov.org/

Примерные Вопросы для контроля

1. Опишите, как вы поняли, что такое машинное обучение (вопрос-дискуссия)
2. Назовите ученого, который считается родоначальником машинного обучения и теории искусственного интеллекта (Алан Тьюринг)

3. Напишите название шахматного суперкомпьютера, который впервые смог обыграть Гарри Каспарова в матче из 6 партий (Deep Blue)
4. Опишите разницу между подходом машинного обучения и традиционным программированием (вопрос-дискуссия)
5. Опишите разницу между обучением с учителем и обучением без учителя (вопрос-дискуссия)
6. Опишите разницу между задачами классификации и задачами регрессии (вопрос-дискуссия)
7. Вас попросят создать программу, которая будет определять кошку или собаку по изображению. К какому типу задач машинного обучения относится эта просьба? (классификация)

2. **Данные**

Список тем, которые должны быть осуждены на лекции:

1. Типы данных
2. Базы данных

Ключевые моменты по темам:

1. Обсуждение подходов к классификации типов данных
    - Категориальные данные (номинальные и порядковые)
    - Числовые данные (дискретные и непрерывные)
    - Табличные данные
    - Изображения
    - Временные ряды
    - Естественный язык

    Рассмотрение примера табличных данных и определение типа данных (дискуссия)

    Обсуждение возможности сведения «других» типов данных к табличным, а также недостатков этого подхода для каждого типа данных.

2. Рассмотрение открытых баз данных для задач машинного обучения. «Классические» базы данных (UCR, UCI) и современные платформы (kaggle, openml и др)

Подведение Итогов Лекции

Текстовый материал

Типы данных

К слову, о признаках и данных. Существуют различные подходы к классификации признаков, назовем их **микроуровень** и **макроуровень**.

На **микроуровне** признаки можно разделить на **Числовые** и **Категориальные**.

Числовые признаки, это как правило количественные оценки. Числовые признаки делят на **дискретные**, которые «невозможно измерить, но можно посчитать», например, ученики в классе, пальцы, результат в футболе; и **непрерывные** данные, которые «не могут быть подсчитаны, но их можно измерить», например, температура, напряжение, высота. Можете отложить на время учебное пособие и придумать другие примеры числовых признаков.

Для обозначения числовых данных как правило используются следующие обозначения:

- $\mathbb{N}$ Натуральные числа
- $\mathbb{Z}$ Целые числа
- $\mathbb{Q}$ Рациональные числа
- $\mathbb{R}$ Действительные числа
- $\mathbb{C}$ Комплексные числа

Категориальные признаки, это как правило характеристики объектов. Обычно категориальные признаки делят на **Номинальные** (**nominal**) признаки, которые отвечают на вопрос о том какое значение принимает данная характеристика, например, цвет, пол, язык, институт и т. д.; и **Порядковые данные (ordinal)** признаки, дискретные и упорядоченные величины, например, уровень английского, итоговая оценка за курс машинного обучения. **Номинальные** признаки, в которых всего два возможных значения называют **бинарными** (как правило это ответы на такие вопросы, когда ожидается ответ «да» или «нет»).

На практике **категориальные** данные могут быть представлены в виде числовых значений, как это представлено на Рис. 2-1.

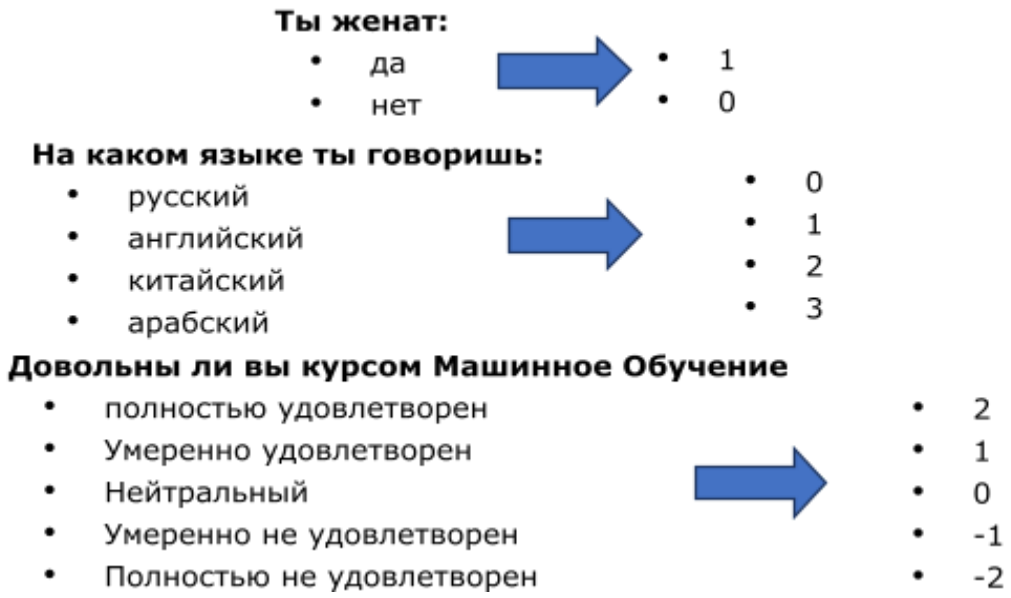


Рис. 2-1 Пример представления категориальных данных в виде числовых данных

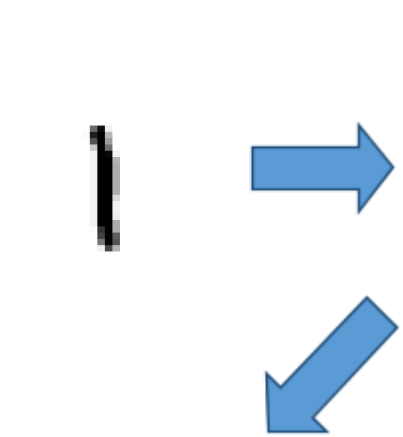
Но нужно помнить, что подобные числа не имеют математического значения, как случае числовых признаков. Т. е. их нельзя складывать (например, нельзя сложить английский и китайский и получить арабский), или сравнивать между собой (английский не в три раза «хуже» чем арабский). Это накладывает определенные ограничения на использование категориальных признаков в моделях машинного обучения.

На **макроуровне** признаки можно разделить на **Табличные** и **другие**. **Табличные** – это данные как на Рис. 1-1, представленные в виде таблице представляющих совокупность различных числовых и категориальных признаков. По строкам представлены различные объекты, по столбцам – различные признаки.

Помимо этого, существуют изображения, временные ряды и естественный язык. Каждый из этих типов данных имеет свои особенности, которые требуют специального подхода. При этом для получения каких-то простых моделей эти данные можно свести к табличным.

Например, можно рассматривать изображение как совокупность интенсивности отдельных пикселей (Рис. 2-2). Принято, что большие значения интенсивности соответствуют белому цвету, а небольшие – черному. При этом в зависимости от разрядности изображения можно иметь разные значения «оттенков серого» между белым и черным. Так на Рис. 2-2 представлено 8-битное изображение (где интенсивность цвета кодируется значением от 0 до 255). Любое изображение можно «спрямить» (flatten), т. е. перейти от двумерного представления к одномерному. Таким образом каждое

изображение можно представить в виде большой строки признаков. Например, изображение 10 на 10 пикселов представляется в виде строки из 100 признаков.



241	251	248	244	252	247	254	254	241	240
244	249	252	246	9	2	240	249	251	252
241	244	254	251	250	8	244	250	254	241
246	251	250	243	249	14	245	246	254	252
244	251	246	244	240	1	244	242	241	249
247	241	244	253	244	15	250	248	249	245
245	252	250	245	248	8	249	250	243	247
254	241	240	249	254	13	247	240	248	242
245	247	241	248	248	1	247	249	252	245
243	244	251	242	248	247	248	251	249	246

241	251	248	244	252	247	254	254	241	240	244	249	252	246	9	2	240	249	251	252

Рис. 2-2 Представление одноканального изображения в виде строки признаков

Однако нужно помнить, что реальные изображения состоят из нескольких цветовых каналов, а не одноканальные как на Рис. 2-2. Наиболее распространена трехканальная цветовая модель RGB – Red, Красный; Green, Зеленый; Blue, Синий. На Рис. 2-3 представлен пример разложения, пожалуй, самого известного изображения в Компьютерном Зрении – Лена – на три канала. Таким образом изображение Лены можно представить в виде строки из $512 \times 512 \times 3 = 786432$ признаков.

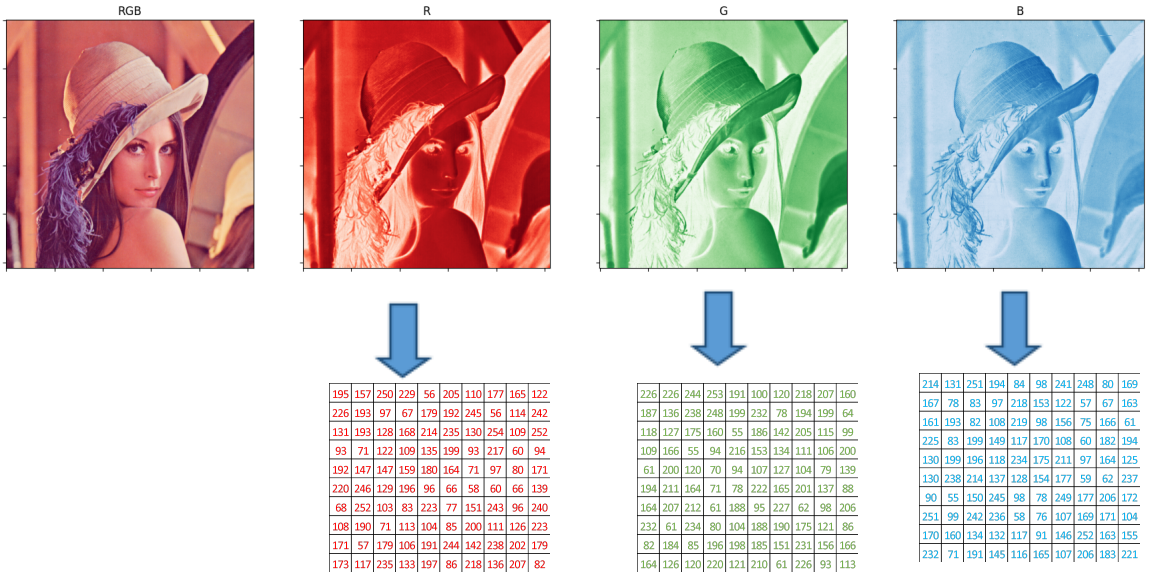


Рис. 2-3 Разложение цветного изображения на каналы

https://en.wikipedia.org/wiki/Lenna#/media/File:Lenna_(test_image).png

Понятно, что такое представление изображение в виде табличных данных не кажется чем-то эффективным. При этом мы должны уверены, что в каждом пикселе находится «однотипная» часть изображения: например, если это изображения котов и собак, то для такого подхода они должны быть сняты под одним углом. Поэтому представление изображений в виде таблиц в основном используется только для однотипных данных (например, изображения цифр). А для обработки более сложных изображений в настоящее время самым распространенным подходом являются модели, использующие Свёрточные Нейронные Сети (Convolutional Neural Networks) [He и др., 2016; Szegedy и др., 2015; Szegedy и др., 2016].

В какой-то степени аналогичным способом можно поступать с временными рядами. Временной ряд представляет собой совокупность значений какого-либо измерения в отдельные моменты времени. С одной стороны, можно свести временной ряд к табличным данным. Однако такой подход наивен, если планируется анализировать сигналы разных объектов – мы должны быть уверены в том, что все сигналы «согласованы» по оси времени.

Однако такое редко встречается для реальных сигналов. Например, сигнал электрокардиографии (Рис. 2-4) имеет достаточно сложную структуру. При этом информативным является расстояние между последовательными R-зубцами.

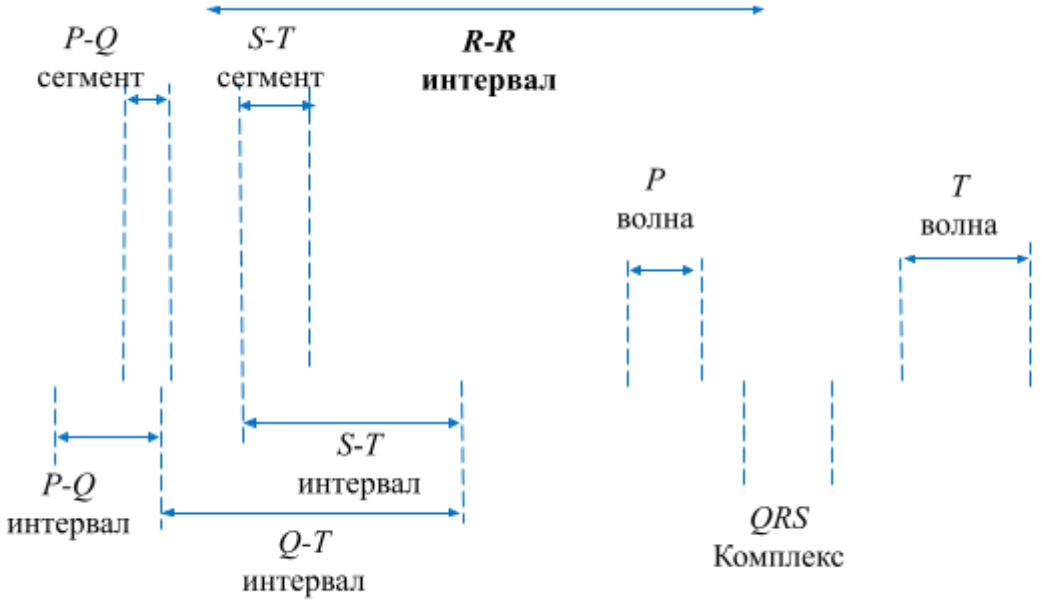


Рис. 2-4 Визуализация сигнала Электрокардиограммы

В этом ключе продуктивным подходом к обработке временных сигналов является расчет признакового пространства. Так подробно обработка биомедицинских сигналов и извлечение признакового пространства освещена в следующих учебных пособиях [Кубланов и др., 2020; Кубланов, Борисов, Долганов, 2016].

С другой стороны, возможно использовать модели, которые учитывают временную составляющую временных рядов. К таким подходам относятся модели, использующие Рекуррентные Нейронные Сети (Recurrent Neural Network) [Hochreiter, Schmidhuber, 1997].

В упрощенном виде данные естественного языка можно рассматривать как совокупность отдельных слов – категориальных признаков. Такие подходы вполне применимы в обобщенных задачах, связанных с оценкой тональности текстов. Однако для задач переводов, генерации текста используют т. н.

модели Трансформеры (Transformers) [Devlin и др., 2019], которые используют механизм Внимание (Attention) [Vaswani и др., 2017].

Ссылки на справочные материалы

1. Платформа используется для обмена открытыми наборами данных, участия в соревнованиях по машинному обучению или обмена кодом в среде Data Science https://www.kaggle.com/
2. Платформа открытого машинного обучения https://www.openml.org
3. Классический репозиторий данных для машинного обучения https://archive.ics.uci.edu/ml/index.php

Примерные Вопросы для контроля

1. Приведите несколько примеров непрерывных и дискретных данных (вопрос-дискуссия)
2. К какому типу данных можно отнести диагноз, поставленный врачом? (категориальные данные)
3. Опишите, какие данные содержатся в наборе данных Iris (https://archive.ics.uci.edu/ml/datasets/Iris)
4. Что делает изображения, тексты на естественном языке и временные ряды особым типом данных? (вопрос-дискуссия)

3. Линейная Алгебра

Список тем, которые должны быть осуждены на лекции:

1. Объекты
2. Операции

Ключевые моменты по темам:

1. Обсуждение основных объектов линейной алгебры
    - Скаляр
    - Вектор
    - Матрица
    - Тензор

    Обозначения основных объектов. Размерности объектов. Рассмотрение примеров объектов линейной алгебры (дискуссия)

2. Основные операции линейной алгебры

    Сложение матриц и умножение матриц. Акцент внимания на отслеживании размерностей объектов при выполнении операций.

    Транспонирование матриц

    Поиск обратной матрицы

    Геометрический смысл матриц. Собственные значения и собственные вектора матриц. Определитель матрицы

Подведение Итогов Лекции

Текстовый материал

Линейная Алгебра

Перед тем, как обучать модель и решать задачи необходимо вспомнить базовые вещи из линейной алгебры для манипуляций с данными. Ключевые вещи, которые необходимо понимать это **Объекты** и **Операции.**

Самым простым объектом является просто число, или **Скаляр**. Обозначается $x \in R$. По-простому – это одна ячейка в табличных данных.

Далее существуют совокупности из нескольких скаляров, которые называются **Векторы**. Обозначаются $x \in R^n$, где n – размерность вектора. В табличных данных выделяют векторы-строки и векторы-столбцы.

Поскольку мы можем объединить несколько скаляров в вектор, то, наверное, мы можем объединить несколько векторов одинаковой размерности в новый объект, который называется **Матрица**. Обозначается как $X = \left[x_{ij}\right]_{m \times n}$ или $X \in R^{m \times n}$, где m – количество строк, n – количество столбцов. По сути, матрицы и есть таблицы данных.

Наконец, мы можем продолжать объединять Матрицы одинаковой размерности в новые объекты, которые называются **Тензоры**. Трехмерный тензор обозначается как $X = x_{ijk}$. Мы с вами уже рассматривали данные в виде тензоров. Это трехканальное цветное изображение.

В Линейной алгебре, по сути, используются те же **Операции**, что и в простой школьной алгебре (сложение, вычитание, умножение, деление), но с некоторыми особенностями. Ключевая особенность – нужно внимательно следить за размерностью объектов, над которыми совершаются операции.

- Сложение матриц

$A, B \in R^{m \times n}; C \in R^{m \times n}$

Результатом сложения двух матриц, у которых одинаковая размерность, является матрица той же размерности, при этом каждый элемент является суммой соответствующих элементов исходных матриц

$$C = A + B$$

- Матрично-скалярное сложение

$A \in R^{m \times n}, x \in R; B \in R^{m \times n}$

При этом мы можем складывать матрицы и скаляры. В данном случае скаляр добавляется к каждому элементу исходной матрицы.

$$B = A + x$$

- Broadcasting (сложение матрицы и вектора)

$A \in R^{m \times n}$, $x \in R^{n}$;$B \in R^{m \times n}$

Также мы можем интересным образом складывать вектора и матрицы. В данном случае количество столбцов в матрице должно совпадать с размерностью вектора. В данном случае вектор распространяется (от английского to broadcast) по всем строкам матрицы

$$B = A + x$$

- Умножение Матрицы на Матрицу

$A = \left[a_{ij}\right]_{m \times n}$ $B = \left[b_{ij}\right]_{n \times p}$; $C = AB = \left[c_{ij}\right]_{m \times p}$

Умножение матрицы на матрицу самое требовательное к размерностям исходных матриц. Мы можем умножать матрицу на матрицу только в том случае, если количество столбцов первой матрицы совпадает с количеством строк второй матрицы. При этом размерность итоговой матрицы, будет следующей: количество строк – будет равно количеству строк первой матрицы, а количество столбцов – количеству столбцов второй матрицы

При этом каждый элемент итоговой матрицы определяется исходя из следующих соображений: $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

При этом в общем случае $AB \neq BA$.

- Поэлементное умножение (произведение Адамара)

$$A = \left[a_{ij}\right]_{m \times n} \quad B = \left[b_{ij}\right]_{m \times n}$$

Стоит также упомянуть операцию поэлементного умножения, которая вычисляется по аналогии с поэлементным сложением. Разумеется, что размерности исходных матриц должны совпадать

$$A \odot B = \left[a_{ij} b_{ij}\right]_{m \times n}$$

- Транспонирование Матрицы

$$A \in R^{m \times n}$$

Иногда возникает необходимость «повернуть» матрицу, т. е. поменять местами столбцы и строки. Такая операция называется транспонированием.

$$A^T \in R^{n \times m}$$

При этом, если мы выполняем операцию транспонирования два раза мы возвращаемся к исходной матрице.

$$A^{TT} = A$$

- «Деление Матриц»

А вот деления матриц не существует. Есть своеобразный аналог делению из школьной алгебры: домножение на Обратную Матрицу. Обратная матрица соответствует следующему условию

$A * A^{-1} = A^{-1} * A = I$, где I - Единичная Матрица

Ссылки на справочные материалы

1. Плейлист с хорошей визуализацией понятий и концепций Линейной Алгебры (на английском) https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab
2. Сайт с интерактивной визуализацией матричного умножения http://matrixmultiplication.xyz/

Примерные Вопросы для контроля

1. Приведите несколько примеров данных в виде тензоров (вопрос-дискуссия)
2. У вас есть три матрицы A, B, C: A имеет размеры 5 × 4, B имеет размеры 4 × 6, C имеет размеры 3 × 5. Напишите все возможные матрицы, которые можно перемножить между собой, и укажите размеры результирующих матриц

   A*B = AB (5x6)

   C*A = CA (3x4)

   CA*B = CAB (3x6)

4. **Основы математического анализа**

Список тем, которые должны быть осуждены на лекции:

1. Функции
2. Производные

Ключевые моменты по темам:

1. Определение функции в математике. Определение функции в программировании

   Способы представления функции: табличная форма, график, формула

   Примеры функций: линейные функции, нелинейные функции.

   Рассмотрение нейронных сетей как совокупность сложных функций
2. Определение производной

   Примеры производных ряда функций

   Таблицы производных: производные степенных функций, экспоненты, логарифма. Правила взятия производных сложных функций: производная произведения, производная линейной комбинации функции, цепное правило

   Градиент – вектор производных. Рассмотрение примеров градиентов разных функций

   Рассмотрение примеров взятия производных сложной функции

Подведение Итогов Лекции

Текстовые материалы

Математический анализ

Итак, мы вспомнили Объекты и базовые Операции. Однако для задач из реального мира требуется более сложное взаимодействие между объектами. Для этого нам понадобятся знания из математического анализа, которые описывают функции.

Функция в математике — соответствие между элементами двух множеств — правило, по которому каждому элементу первого множества соответствует

один и только один элемент второго множества. Схожая аналогия существуют также и, например, в программировании.

Самый простой пример функции – функция, которая ничего не делает. Эта функция берет на вход переменную x и возвращает ее. Функции можно представить по-разному, можно в виде математического выражения: $f(x) = x$, а можно в виде графика функции (Рис. 1-1 а). Как видно данная функция выглядит как прямая линия. При этом у функции могут быть дополнительные параметры, например входные данные можно умножить на скаляр и добавить скаляр $f(x) = 0.5 * x - 1$, график такой функции представлен на (Рис. 1-1 б). График все еще выглядит как прямая линия. Такие функции называются линейными функциями, т. е. выходной аргумент функции пропорционален входному аргументу.

Функции бывают также и нелинейными. Например, в раннее время использования Нейронных Сетей была достаточно распространена сигмоидная функция, которую также называют логистической, которая выражается уравнением $f(x) = \frac{1}{1+e^{-x}}$ (Рис. 1-1 в). Другим примером нелинейной функции является функция ReLU (Rectified linear unit – линейный выпрямитель) $f(x) = (0, x)$ (Рис. 1-1 г).

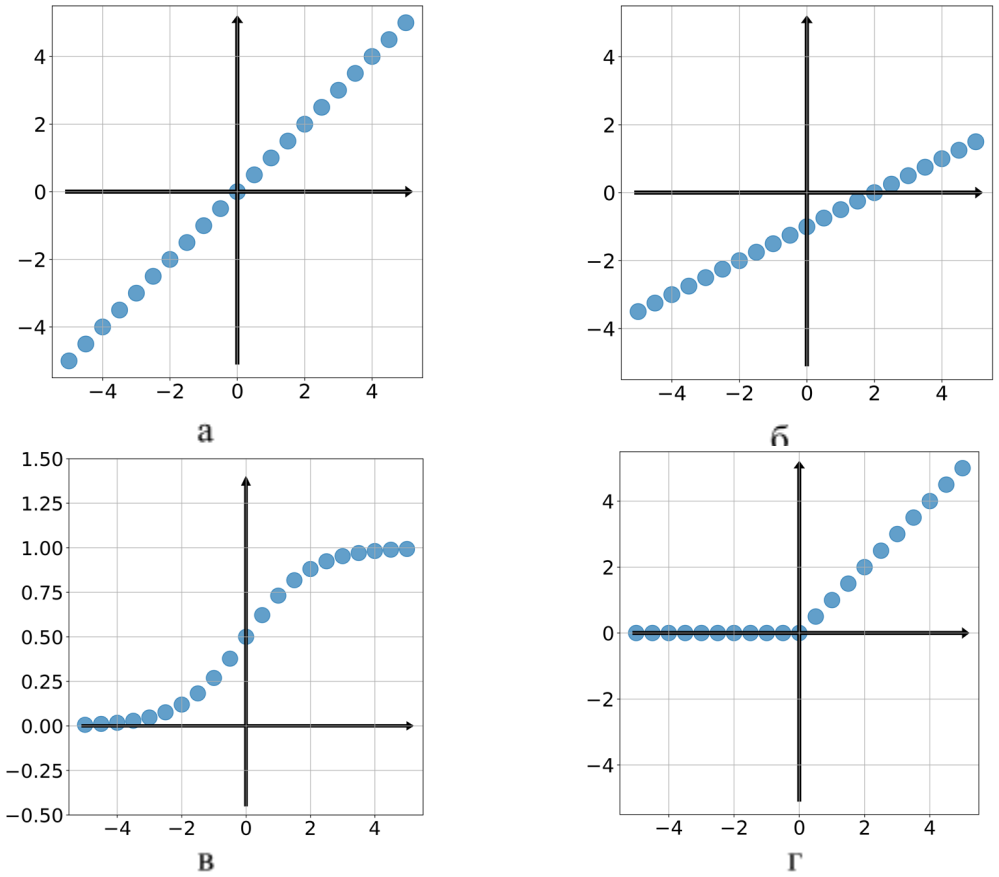


Рис. 4-1 Примеры графиков функций

Для анализа поведения функций часто используют другую функцию, которая называется производной. **Производная** функции в точке — скорость изменения функции в данной точке. Производную можно определить, как предел отношения приращения функции к приращению аргумента.

$$f'(x_0) = \frac{f(x) - f(x_0)}{x - x_0} = \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} = \frac{\Delta f(x)}{\Delta x}$$

На Рис. 4-2 представлены примеры производных для линейной функции (а), логистической функции (б) и ReLU (в). Так видно, что скорость изменения линейной функции – постоянна, логистической функции зависит от области значений, а для функции ReLU меняется «скачкообразно».

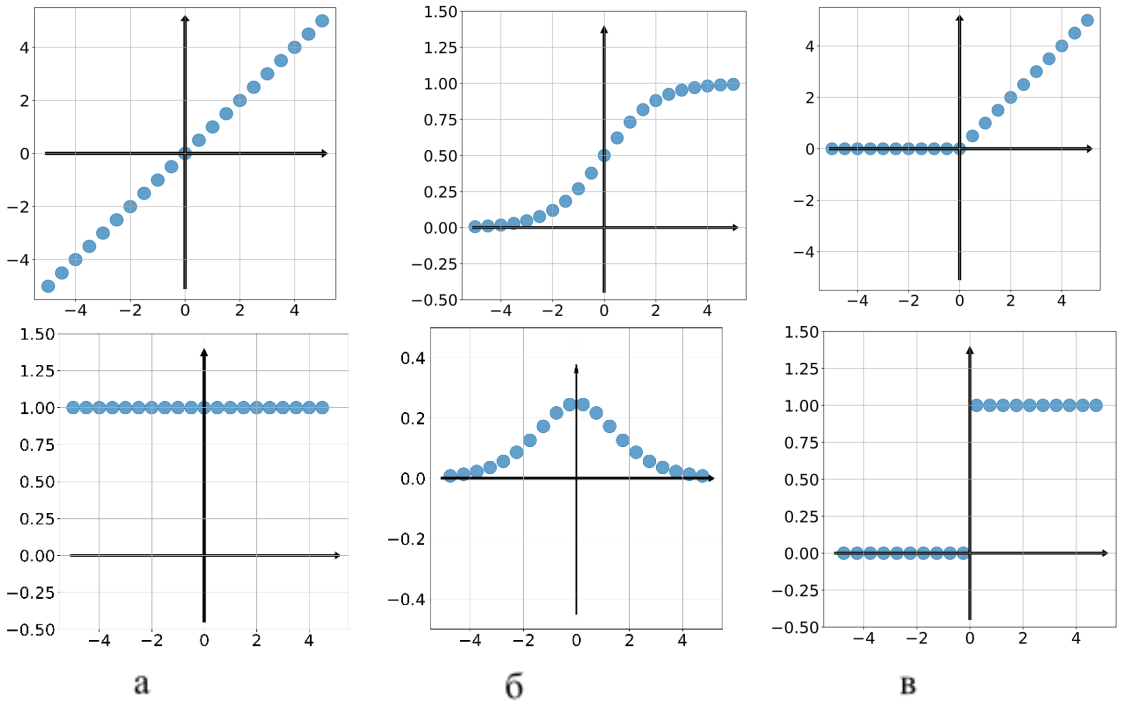


Рис. 4-2 Примеры функций и их производных

В таблице 1 представлены функции и их производные, которых достаточно знать для успешного усвоения материала данного учебного пособия. Отдельно стоит упомянуть последнюю строку в таблице – т. н. цепное правило. Если y это функция от переменной t, а t в свою очередь зависит от переменной x, тогда производная y по переменной t будет равна производной y по переменной t помноженную на производную t по переменной x.

Таблица 1 Функции и Производные

Функция	**Производная**
c - константа	0
x	1
x^m	$m * x^{m-1}$
$log(x)$	$\frac{1}{x}$
e^x	e^x
$f(x) * g(x)$	$f'(x) * g(x) + f(x) * g'(x)$

$c * f(x)$	$c * f'(x)$
$y = f(t)$ $t = g(x)$	$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial t}\frac{\partial t}{\partial x}$

Функции могут зависить от разных переменных. Например, $f(x, w, b) = wx + b$. Мы можем посчитать производные по трем переменным x, w, b. Если мы считаем производную $\frac{\partial f}{\partial x}$ то остальные переменные считаются константами. Производные по отдельным переменным называются частными производными. Вектор, составленный из частных производных, называется **градиент**. Для упомянутой выше функции будет выглядеть следующим образом $\nabla f(x, w, b) = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial w}, \frac{\partial f}{\partial b}] = [w, x, 1]$.

Отдельно рассмотрим производную квадратичной функции $f(x) = x^2$. Напомним, что мы определили, что в общем случае цель обучения, это минимизация функционала потерь $Q(a, X) \rightarrow$. Допустим у этого функционала всего 1 параметр w, а сама функция потерь Q квадратично зависит от этого параметра (Рис. 4-3).

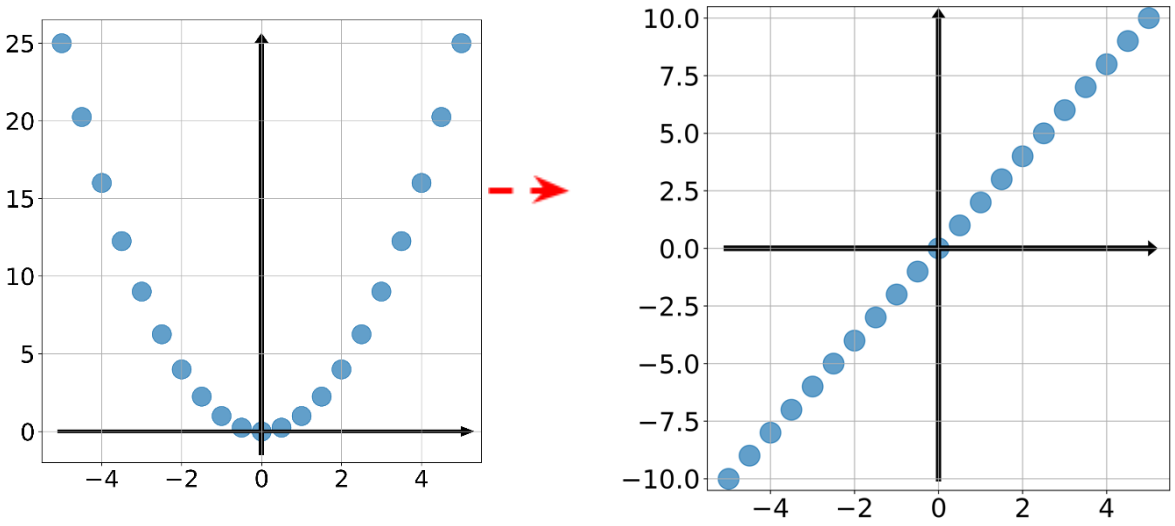


Рис. 4-3 Квадратичная функция и ее производная

Для того чтобы найти минимум этой функции нужно приравнять производную этой функции к нулю. Значение параметра w_0, при котором

производная $\frac{dQ(w=w_0)}{dw}$ равна нулю будет соответствовать параметру, который минимизирует исходный функционал. Запомним это.

С другой стороны, допустим мы подобрали значения параметра случайным образом и хотим оценить, насколько хорош этот параметр и можно ли его как-то изменить, используя производную. Допустим $\frac{dQ(w=-2)}{dw} = -4$. Что это значит? В этой точки функция убывает (производная отрицательная), т. е. если продолжать увеличивать параметр, то мы скорее всего придем к минимуму функции. Аналогично рассмотрим $\frac{dQ(w=4)}{dw} = 8$. Производная положительная, а значит функция возрастает. Причем возрастает быстрее, чем в точке $w = -2$. Это значит, что если мы уменьшим параметр, то мы приблизимся к минимуму функции. При этом нам нужно сделать больший шаг, поскольку абсолютная величина производной больше. Эта идея изменения параметров в направлении обратной знаку производной на величину пропорциональную значению производной и легла в основу алгоритма градиентного спуска

Ссылки на справочные материалы

1. Плейлист с хорошей визуализацией понятий и концепций Математического Анализа (на английском) https://www.youtube.com/playlist?list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr
2. Сводная информация о производной функции https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8

5. Предварительная Обработка

Список тем, которые должны быть осуждены на лекции:

1. Начальные Шаги Предварительной Обработки
2. Типы Предварительной Обработки
3. Базовая Генерация Признаков

Ключевые моменты по темам:

1. Рассмотрение возможностей библиотеки Pandas для анализа данных. Объект датафрейм (dataframe).

   Методы библиотеки Pandas для поиска пропусков (.isna()), заполнения пропусков (.fillna(value)), поиска дубликатов (.duplicated()), удаления дубликатов (.drop_duplicates()).

   Применение агрегации для первичного анализа данных (.groupby())

   Особенности применения библиотеки seaborn для визуализации датафреймов Pandas
2. Основные типы предварительной обработки данных

   Мотивация для предварительной обработки данных.

   Стандартизация и нормализация для предварительной обработки данных (линейные преобразования которые не изменяют распределение данных)

   Анализ выбросов при предварительной обработке

   Степенное преобразование (нелинейное преобразование для получения нормального распределения)
3. Простые подходы к генерации дополнительных признаков.

   One-hot encoding для использования категориальных признаков в линейных моделях. Метод .get_dummies библиотеки Pandas для реализации One-hot encoding.

   Анализ распределения категориальных признаков. Редкие категории.

Категориальное сопоставление. Особенности методов .cut() и .qcut() библиотеки Pandas для разбивки значений на дискретные интервалы и дискретизации на основе квантилей.

Комбинации категориальных признаков.

Обсуждение возможных источников для новых признаков:

Знания, специфичные для предметной области (чтение статей, взаимодействие с научным руководителем и/или экспертом в данной предметной области и т.д.)

Исследовательский анализ данных (статистический анализ данных, анализ распределений данных)

Подведение Итогов Лекции

Текстовые материалы

В качестве набора данных мы подобрали набор Car Moldova - Оценка стоимости и типа трансмиссии по данным продаж автомобилей на вторичном рынке Молдавии. Набор данных представляет собой статистику параметров автомобилей на вторичном рынке в Молдавии. Набор включает ряд категориальных и численных значений, составляющих одну запись (строку). Каждый столбец в записи — это отдельный признак. Среди указанных признаков приведены целевой для задачи предсказания (регрессии) - цена автомобиля. Также среди параметров есть целевой для задачи классификации - тип трансмиссии. Последняя задача может быть рассмотрена, например, как пример задачи на заполнение пропусков (если продавец не указал соответствующий параметр).

Для начального анализа данных крайне рекомендуется использовать библиотеку Pandas [9].

Библиотека Pandas для анализа данных

Для начала работы с любой библиотекой необходимо импортировать её

```
import pandas as pd
```

Теперь давайте загрузим данные в структуру Pandas датафрейм (Dataframe). Эта структура представляет собой двумерную структуру данных с именными столбцами потенциально разных типов. Вы можете думать об этом как о файле Excel, но на языке Python.

Считывание файлов в датафрейм

Для загрузки данных мы можем использовать функцию pd.read_csv(path,delimiter). Для успешного использования необходимо указать путь (path) к файлу. В общем случае путь это строка, содержащий полный путь к файлу и название файла.

Мы также можем указать разделитель (delimiter), в нашем случае это «,». Но для разных файлов могут использоваться и другие типы разделителей, такие как «;», « » (пробел), «\t» (табуляция). Вы можете проверить используемый в файле разделитель, предварительно открыв его в текстовом редакторе, например Notepad++.

Так следующая строка загрузит файл с названием cars_moldova.csv, который лежит в папке /content/, разделитель – запятая.

```
df = pd.read_csv('/content/cars_moldova.csv', delimiter = ',')
```

Для того чтобы визуализировать содержание загруженного датафрейма в блокнотах Google Colab достаточно просто запустить отдельную ячейку, в которой прописана переменная, которая содержит датафрейм. В других случаях можно воспользоваться функцией display. На представлена Рис. 5-1 визуализация датафрейма набора Car Moldova

	Make	Model	Year	Style	Distance	Engine_capacity(cm3)	Fuel_type	Transmission	Price(euro)
0	Toyota	Prius	2011	Hatchback	195000.0	1800.0	Hybrid	Automatic	7750.0
1	Renault	Grand Scenic	2014	Universal	135000.0	1500.0	Diesel	Manual	8550.0
2	Volkswagen	Golf	1998	Hatchback	1.0	1400.0	Petrol	Manual	2200.0
3	Renault	Laguna	2012	Universal	110000.0	1500.0	Diesel	Manual	6550.0
4	Opel	Astra	2006	Universal	200000.0	1600.0	Metan/Propan	Manual	4100.0
...	...	...	...	...	...	...	...	...	...
41002	Dacia	Logan Mcv	2015	Universal	89000.0	1500.0	Diesel	Manual	7000.0
41003	Renault	Modus	2009	Hatchback	225.0	1500.0	Diesel	Manual	4500.0
41004	Mercedes	E Class	2016	Sedan	50000.0	1950.0	Diesel	Automatic	29500.0
41005	Mazda	6	2006	Combi	370000.0	2000.0	Diesel	Manual	4000.0
41006	Renault	Grand Scenic	2006	Minivan	300000.0	1500.0	Diesel	Manual	4000.0

41007 rows × 9 columns

Рис. 5-1 Визуализация датафрейма набора Car Moldova

Общая информация о датафрейме

Чтобы получить общую информацию о содержимом датафрейма. Для этого мы просто применяем метод .info() к нашему датафрейму. Вы должны увидеть следующую информацию (Рис. 5-2): количество столбцов, имена столбцов, тип данных в каждом столбце, количество пропущенных значений для каждого столбца.

```
RangeIndex: 41007 entries, 0 to 41006
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Make                  41007 non-null  object
 1   Model                 41007 non-null  object
 2   Year                  41007 non-null  int64
 3   Style                 41007 non-null  object
 4   Distance              41007 non-null  float64
 5   Engine_capacity(cm3)  41007 non-null  float64
 6   Fuel_type             41007 non-null  object
 7   Transmission          41007 non-null  object
 8   Price(euro)           41007 non-null  float64
dtypes: float64(3), int64(1), object(5)
memory usage: 2.8+ MB
```

Рис. 5-2 Информация о датафрейме набора Car Moldova

Поиск и удаление Дубликатов

Давайте разберемся с повторяющимися значениями в наборе данныъ. Повторяющиеся строки могут возникать по многим причинам: программная ошибка, человеческая ошибка и т. д. При этом никакой пользы от включения повторных строк не будет для алгоритмов машинного обучения.

Вы можете проверить количество повторяющихся строк, применив метод .duplicated() к датафрейму. В результате получится столбец, который содержит информацию о том, является ли данная строка повторной или нет. Чтобы узнать количество повторных строк мы можем применить метод .sum()

```
df.duplicated().sum()
```

Для датафрейма набора Car Moldova имеется 3743 дубликата, почти 1/10 от всех данных. Чтобы удалить дубликаты, мы просто используем следующую строку

```
DF = df.drop_duplicates().reset_index(drop=True)
```

Позвольте мне объяснить, что здесь происходит:

- мы берем наш оригинальный датафрейм df;
- мы просим Python использовать метод удаления дубликатов .drop_duplicates();

- мы просим сбросить индексы датафрейма .reset_index(drop=True), в качестве альтернативы вы можете сохранить исходные индексы датафрейма;
- мы сохраняем результаты вышеупомянутого преобразования в новый датафрейм DF .

Хорошей практикой является размещение результата значительного преобразования в отдельном датафрейме, чтобы вы всегда могли вернуться к исходному датафрейму. Вы можете проверить, что все сделано правильно, оценив количество строк дубликатов в новом датафрейме.

Сохранение датафрейма в файл

Для сохранения датафрейма в файл можно воспользоваться методом .to_csv(path, index), которому необходимо указать полный путь и название файла, в который вы хотите сохранить данные (path). Также можно указать, необходимо ли сохранять индексы строк (булева переменная index). Так следующая строка сохранит датафрейм без дубликатов в папку /content/ с названием 'cars_moldova_no_dup.csv'

```
DF.to_csv('/content/cars_moldova_no_dup.csv',index=False)
```

Представление части датафрейма

Здесь стоит упомянуть два методах: .head(n) и .tail(n). Эти два метода можно применить к нашему датафрейму для визуализации первых n или последних n строк. Это очень полезно, когда ваш датафрейм довольно большой и не может быть легко визуализирован полностью. Попробуйте сами: покажите первые 6, а затем последние 9 строки нашего датафрейма.

Индексация

Теперь поговорим об индексации. Одним из способов получения определенных элементов датафрейма является использование атрибута .loc

Ниже приведены некоторые из примеров

• взятие одной ячейки DF.loc[1437,'Transmission']

Этот код элемент из строки с индексом 1437 в столбце 'Transmission'.

• взятие одной колонки в формате серий (Series) DF.loc[:,'Transmission']

Этот код вернет все элементы в столбце 'Transmission'.

• взятие одной колонки в формате датафрейма DF.loc[:,['Transmission']]

• взятие нескольких колонок DF.loc[:,['Transmission','Year']]

Этот код вернет все элементы в столбцах 'Transmission' и 'Year'.

• взятие нескольких колонок подряд (среза столбцов) DF.loc[:,'Make':'Style']

Этот код вернет все элементы в столбцах, начиная с 'Make' по 'Style' (включительно)

Теперь поговорим о построчной индексации.

• взятие одной конкретной строки в формате серий DF.loc[69,:]

Этот код вернет все элементы из строки с индексом 69.

• взятие одной конкретной строки в формате датафрейма DF.loc[69:69,:]

• получение среза строк DF.loc[322:1437,:]

Этот код вернет все элементы в строках, начиная с индекса 322 и заканчивая индексом 1437 (включительно)

И, конечно же, вы можете комбинировать срезы по строкам и срезы по столбцам DF.loc[227:229,'Make':'Fuel_type']. Этот код вернет все элементы в строках, начиная с индекса 227 и заканчивая индексом 229 (включительно), и в столбцах с 'Make' по 'Fuel_type'.

Альтернативой использования атрибута .loc является использование атрибута .iloc. Основное различие между .loc и .iloc заключается в следующем: .loc работает с названиями столбцов, а .iloc использует вместо этого целочисленную нумерацию.

Так же доступно логическое индексирование (Boolean Indexing) когда мы хотим взять такую часть датафрейма, которая соответствует некому условию. Например строка DF[DF['Transmission']=='Manual'] вернет все элементы исходного датафрейма, для которых выполняется условие что признак в столбце 'Transmission' равен 'Manual'.

Некоторые функции не могут работать с датафреймами напрямую, а рассчитаны на то, что данные подаются в формате массивов, например numpy array. Чтобы перейти от датафреймов к массивам достаточно воспользоваться методом .values к необходимой части датафрейма.

Сортировка DataFrame

Для сортировки данных в датафрейме можно воспользоваться методом .sort_values. Вам также нужно указать столбец, по которому вы хотите отсортировать.

Например строка DF.sort_values(by = 'Price(euro)') отсортирует строки датафрейма по значениям столбца 'Price(euro)'. По умолчанию метод .sort_values сортирует значения по возрастанию. Кроме того, вы можете указать «направление» сортировки, используя переменную ascending. Например, приведенный ниже код сортирует по убыванию столбца 'Year'

```
DF.sort_values(by = 'Year', ascending= False)
```

Визуализация данных

Для визуализации данных можно использовать «стандартную» для Python библиотеку визуализации matplotlib [10; 11]. Однако при использовании датафреймов Pandas целесообразно пользоваться библиотекой Seaborn [12; 13]. Seaborn — это библиотека для создания разнообразной графики на Python, которая тесно интегрирована со структурами данных pandas.

```
import seaborn as sns
```

Один из лучших графиков для ознакомления с данными — это парный график (pairplot). Этот график отображает все возможные попарные комбинации числовых признаков наборах данных в виде скаттерограмм, а по диагонали строятся гистограммы распределений. Но будьте осторожны, если у вас много признаков, это может занять много времени, и может быть лучше разделить датафрейме на несколько частей. Категориальные признаки можно визуализировать используя, например, цвет данных. При этом seaborn позволяет реализовать визуализацию достаточно просто, буквально в одну строчку:

```
sns.pairplot(data = DF, hue ='Transmission')
```

Для того что построить график на достаточно указать датафрейм из которого будут взяты численные признаки в переменную data, дополнительно можно указать в качестве переменной hue (цвет) один из столбцов с категориальными данными. На Рис. 5-3 представлен pairplot для набора данных Cars Moldova.

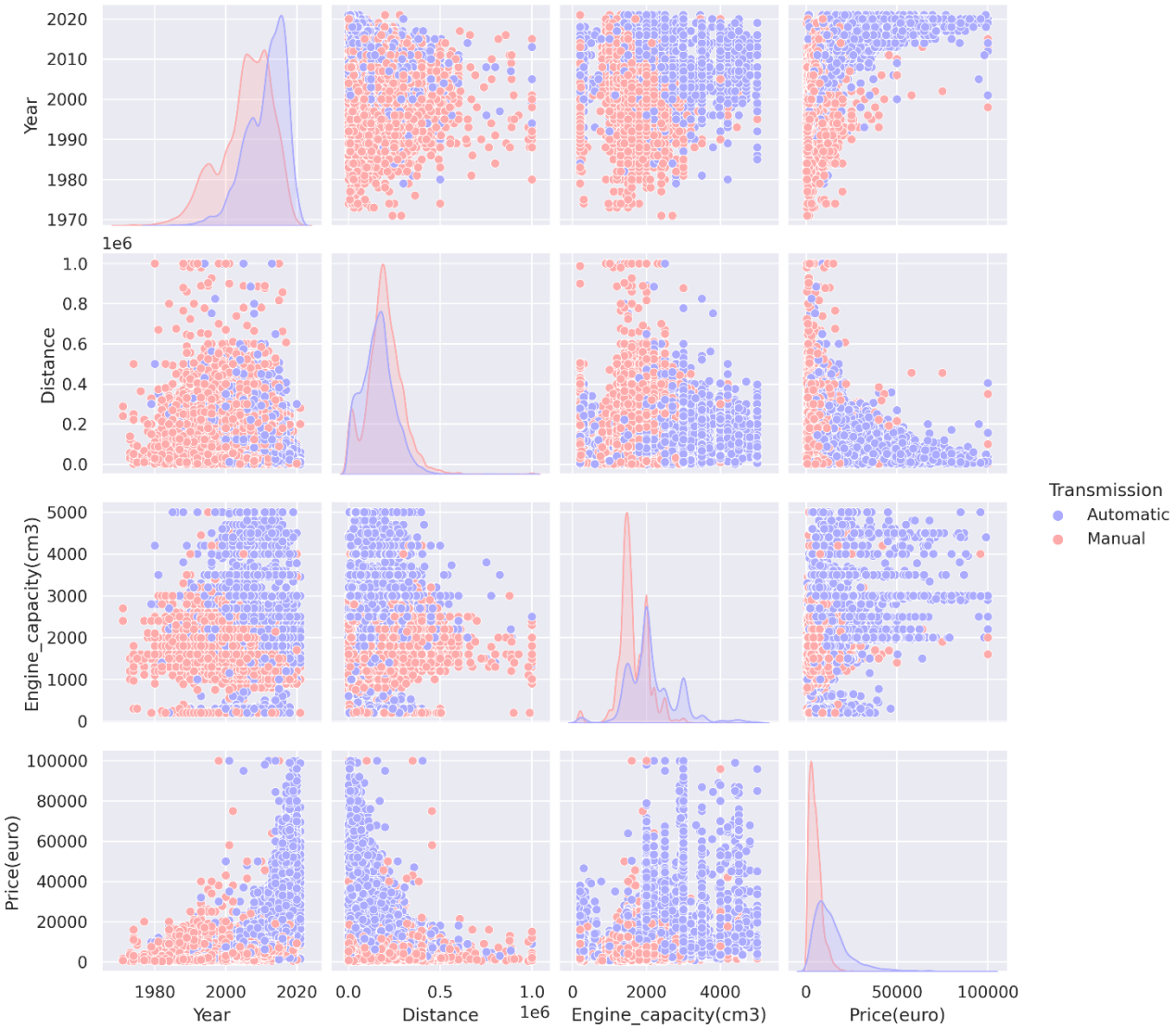


Рис. 5-3 Визуализация pairplot для набора данных Cars Moldova

Возможности библиотеки Seaborn для визуализации данных ограничены только вашим воображением. Ознакомиться с другими визуализациями Seaborn по следующей ссылке [14].

Предварительная Обработка Данных

После ознакомления с данными необходимо выполнить предварительную обработку данных. Эта обработка зависит от типа данных. Для того, чтобы определить типы данных можно воспользоваться следующим кодом

```
cat_columns = []

num_columns = []
```

```
for column_name in df.columns:
    if (df[column_name].dtypes == object):
        cat_columns +=[column_name]
    else:
        num_columns +=[column_name]
```

В переменную cat_columns попадут названия все колонок с категориальными признаками, а в num_columns – с числовыми признаками. Обсудим методы предварительной обработки характерные для разных типов данных.

Предварительная обработка числовых данных

Начать анализ числовых данных стоит с оценки статистических показателей. В библиотеке Pandas это можно сделать в одну строчку с использованием метода .describe()

```
DF[num_columns].describe()
```

На Рис. 5-4 представлен результат применения метода .describe() к набору данных Cars Moldova

	Year	Distance	Engine_capacity(cm3)	Price(euro)
count	37264.000000	3.726400e+04	37264.000000	3.726400e+04
mean	2007.709264	4.758488e+05	1858.932535	9.569387e+03
std	8.295806	4.591520e+06	707.662731	5.283315e+04
min	1900.000000	0.000000e+00	0.000000	1.000000e+00
25%	2004.000000	9.000000e+04	1499.000000	3.300000e+03
50%	2009.000000	1.700000e+05	1800.000000	6.490000e+03
75%	2014.000000	2.300000e+05	2000.000000	1.179900e+04
max	2021.000000	1.000000e+08	9999.000000	1.000000e+07

Рис. 5-4 Результат применения метода .describe() к числовым признакам набора данных Cars Moldova

В результате демонстрируются следующие оценки: количество данных в столбце (**count**), среднее (**mean**), стандартное отклонение (**std**), минимальное значение (**min**), 25, 50 и 75 перцентили (**25%, 50%, 75%**), максимальное значение (**max**).

Сопоставление среднего значения, минимальных и максимальных значений, а также перцентилей позволяет оценить, а существуют ли в наших данных аномалии – редко встречающиеся значения. Также для этого пригодится визуализация гистограмм распределения. Иногда, как например для колонки 'Price(euro)' числовые признаки могут изменяться в большом диапазоне: от 1 до 10^7 , хотя среднее значение, 25 и 75 перцентили сконцентрированы в диапазоне 10^3 ... 10^4. Тогда для визуализации гистограммы распределения рекомендуется использовать логарифмический масштаб. Это говорит, как о аномально высоких и аномально низких значениях, которые встречаются крайне редко. А если какие-то признаки встречаются крайне редко, то на таких значениях сложно обучить модель с высокой степенью обобщения. Поэтому как правило от редких высоких и низких значений признаков избавляются.

При этом аномалии могут быть вызваны ошибками при заполнении данных. Так ошибочными выглядят значения столбца стоимость 'Price(euro)' меньше 100, или старые автомобили с общим пробегом меньше, чем 1000 км.

В датафреймах Pandas это можно реализовать с использованием метода .drop и логической индексации. Ниже представлены рекомендуемые условия по удалению аномальных значений

```
# Старые автомобили с низким пробегом
question_dist_year = DF[(DF.Year <2021) & (DF.Distance < 1100)]
DF = DF.drop(question_dist_year.index)
# Аномально большой пробег
question_dist = DF[(DF.Distance > 0.5e6)]
DF = DF.drop(question_dist.index)
# Слишком малые значения объема двигателя
question_engine = DF [DF["Engine_capacity(cm3)"] < 200]
DF = DF.drop(question_engine.index)
# Слишком большие значения объема двигателя
question_engine = DF[DF["Engine_capacity(cm3)"] > 5000]
DF = DF.drop(question_engine.index)
# Аномально низкие цены
question_price = DF[(DF["Price(euro)"] < 101]
DF = DF.drop(question_price.index)
# Слишком дорогие автомобили, которых мало
question_price = DF[DF["Price(euro)"] > 1e5]
DF = DF.drop(question_price.index)
#Слишком старые автомобили, которых мало
```

```
question_year = DF[DF.Year < 1971]

DF= DF.drop(question_year.index)
```

С другой стороны видно что разные столбцы имеют разный разброс данных: год и объем двигателя измеряется в тысячах, стоимость измеряется в десятках тысяч, а пробег в сотнях тысяч. Интуитивно проще сопоставлять данные, если они измеряются в одних диапазонах. Для приведения численных данных к единой шкале существуют различные подходы: стандартизация, нормализация, степенное преобразование.

Стандартизация

Стандартизация, или z-нормировка, сводится к вычитанию из матрицы признаков $X \in R^{n \times p}$ вектора μ_j средних значений для каждого признака, и делению на вектор σ_j стандартного отклонений для каждого признаков

$$X^{'} = \frac{(x_{ij} - \mu_j)}{\sigma_j}$$

Таким образом у новой матрицы признаков будет нулевое среднее и единичная дисперсия. При этом стандартизация – линейная операция. Т.е. распределение признаков не изменится. Изменится только масштаб в рамках которых это изменение происходит.

Чтобы реализовать стандартизацию на датафреймах Pandas достаточно применить методы .mean() и .std() для оценки среднего значения M и стандартного отклонения STD для каждого столбца.

```
M = DF[num_columns].mean()

STD = DF[num_columns].std()
```

Затем из исходного датафрейма происходит поэлементное вычитание и деление.

DF_scaled = (DF[num_columns]-M)/STD

На Рис. 5-5 представлено распределение признака Distance до и после операции стандартизации. Как видно из Рис. 5-5 распределение данных, остается тем же самым. Меняются границы диапазона измерений.

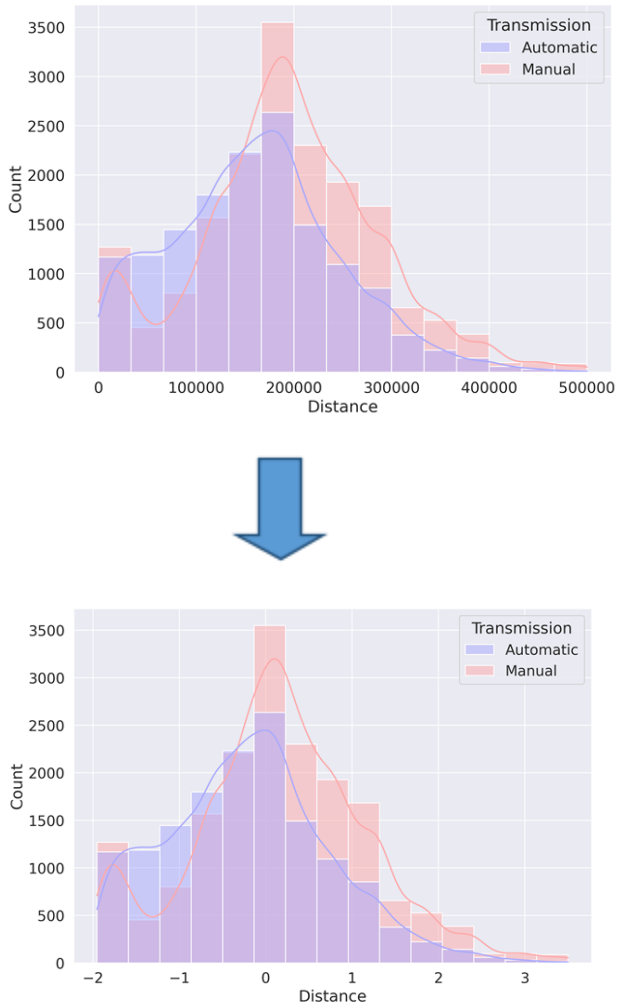


Рис. 5-5 Результат применения стандартизации к столбцу Distance набора данных Cars Moldova

Отдельно стоит отметить, что при выполнении стандартизации (как и других методов предварительной обработки) необходимо сохранять параметры преобразования, поскольку именно эти преобразования необходимо совершать на новых данных. Т. е. для тестового набора данных нужно вычитать среднее не тестового набора, а среднее тренировочного набора.

Нормализация

Нормализация — это тоже линейное преобразование численных признаков $X \in R^{n \times p}$. Но в отличие от стандартизации нормализация переводит распределение с интервал от 0 до 1.

$$X' = \frac{(x_{ij} - x_{j_{min}})}{(x_{j_{max}} - x_{j_{min}})}$$

Как и в случае стандартизация нормализация также просто реализуется с использованием датафреймов Pandas. Используются методы .min() и .max() для поиска минимальных и максимальных значений по столбцам.

```
Xmin = DF[num_columns].min()

Xmax = DF[num_columns].max()
```

Затем аналогично из исходного датафрейма происходит поэлементное вычитание и деление.

```
DF_norm = (DF[num_columns]- Xmin)/( Xmax - Xmin)
```

На рисунке Рис. 5-6 представлено распределение признака Distance до и после операции нормализации. Аналогично, распределение данных, остается тем же самым. Меняются границы диапазона измерений.

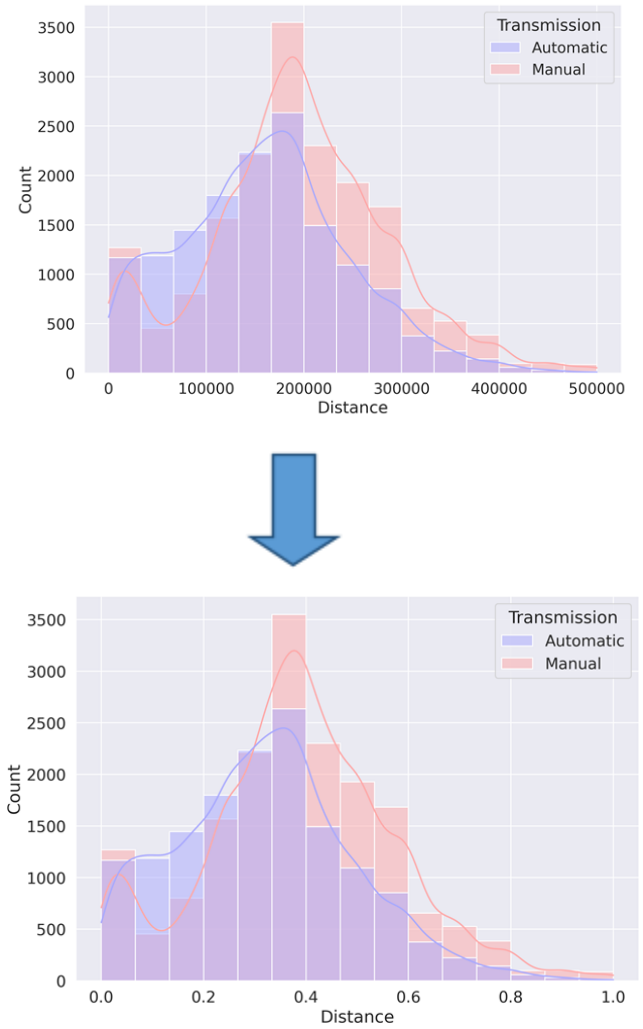


Рис. 5-6 Результат применения нормализации к столбцу Distance набора данных Cars Moldova

Степенное Преобразование

Стоит отметить, что корректное применение преобразований стандартизации и нормализации требует нормального распределения числовых данных. Однако такое происходит не всегда. Тогда рекомендуется предварительно использовать нелинейные преобразования (Power Transform), и уже потом результат нелинейного преобразования стандартизировать или нормализовать. К нелинейным преобразованиям относят логарифмирование

и степенные преобразования (как правило это корни, т.е. степени меньше единицы).

Так на Рис. 5-7 представлено распределение признака Price(euro) до и после операции степенного преобразования. В конкретно этом примере мы сначала логарифмировали столбец, а потом применили стандартизацию.

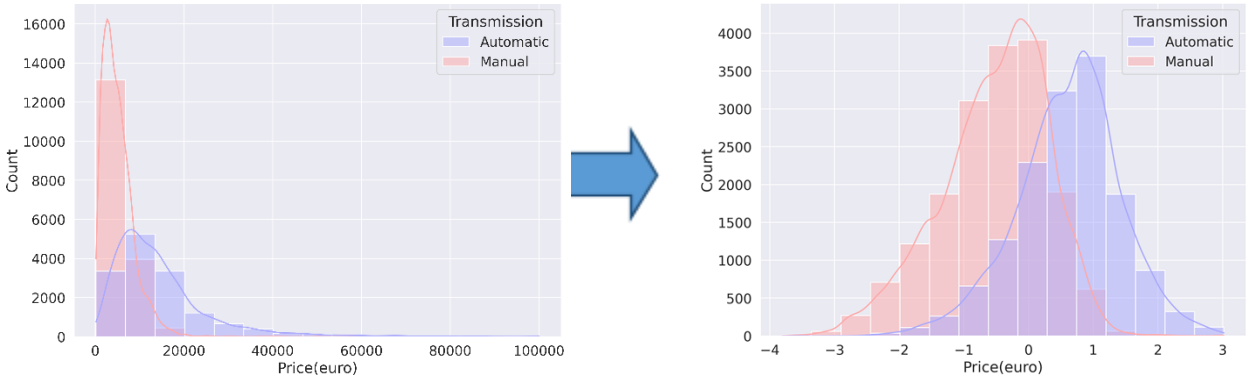


Рис. 5-7 Результат применения степенного преобразования к столбцу Price(euro) набора данных Cars Moldova

Стоит подчеркнуть, что заранее нельзя узнать какой из типов предварительной обработки данных лучше скажется на предсказаниях модели. Поэтому тип предварительной обработки данных можно рассматривать как дополнительный гиперпараметр модели.

Предварительная обработка категориальных данных

Анализ категориальных данных начинается с оценки частоты встречаемости отдельных категорий в рамках столбцов. Первым этапом необходимо воспользоваться методом .nunique() и оценить количество уникальных значений для каждой категории.

```
DF[cat_columns].nunique()
```

Для набора данных Cars Moldova столбцы Make и Model имеют 78 и 777 уникальных значений, соответственно.

Далее необходимо оценить частоту встречаемости уникальных значений с использованием метода .value_counts()

```
counts = DF.Make.value_counts()
```

Можно заметить, что для столбца Make, порядка 15 уникальных значений встречаются 10 и меньше раз. Для выборки из ~30000 объектов это слишком мало. Но в отличии от численных признаков, для которых мы удаляли редко встречающиеся значения, для категориальных признаков мы можем сделать замену на единый класс – редкий (rare). Для этого мы воспользуемся логическим индексированием и методом .replace()

```
rare =  counts[(counts.values < 25)]

DF['Make'] = DF['Make'].replace(rare.index.values, 'Rare')
```

Приведение категориальных признаков к числовым

Далее стоит обсудить формат хранения категориальных данных. В наборе данных Cars Moldova категориальные данные представляют собой тип данных object, и по сути являются строками. Для уменьшения объема хранимой информации достаточно часто эти данные переводят в числовой формат.

Если уникальных значений не много, то такой перевод удобно реализовать с помощью метода .map() и словаря. Так в строке ниже мы преобразуем столбец Transmission, в котором всего 2 уникальных значения.

```
DF['Transmission'] = DF['Transmission'].map({'Automatic': 1, 'Manual': 0})
```

Однако если уникальных значений много, то такое преобразование может быть затруднено. Поэтому рекомендуется воспользоваться изменением типа данных от object к category. А затем кодирование отдельных столбцов в числовые значения с помощью .cat.codes. Полный пример перехода к числовым значениям представлен ниже.

```
DF_ce = df.copy()
```

```
DF_ce[cat_columns] = DF_ce[cat_columns].astype('category')


for _, column_name in enumerate(cat_columns):
    DF_ce[column_name] =  DF_ce[column_name].cat.codes
```

Такое кодирование называют порядковым кодированием (Ordinal Encoding). Стоит отметить, что порядковое кодирование позволило сохранить объем данных примерно на треть: с 2.2 MB до 1.4 MB.

One-hot Кодирование

Существуют некоторые алгоритмы, которые способны работать с категориальными данными с порядковым кодированием. Это естественное кодирование порядковых переменных. Для категориальных переменных он налагает порядковое отношение там, где такого отношения может не быть. В общем случае использование этого кодирования и разрешение модели принимать естественное упорядочение между категориями может привести к снижению производительности или неожиданным результатам (прогнозы на полпути между категориями).

Достаточно распространено использование One-hot кодирование (Encoding). Суть этого кодирование состоит в том, что вместо одного вектора $x^{n\times1}$ категориального признака создается матрица $\Xi^{n\times m}$, где m – количество уникальных категорий в векторе x. При этом матрица Ξ состоит из только 0 и 1. Пример перевода категориального признака с использованием One-hot Encoding представлен на Рис. 5-8.

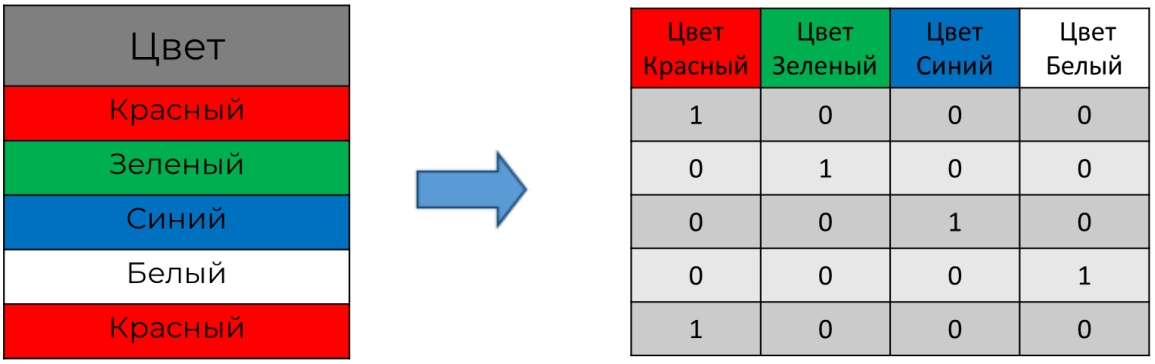


Рис. 5-8 Пример реализации One-hot кодирования для признака Цвет

Категориальный признак «цвет» состоит из 4 уникальных категорий. В результате One-hot кодирование создается 4 новых столбца, по одному на каждое уникальное значение цвета (красный, зеленый, синий, белый). При этом эти столбцы заполнятся в основном 0, а для тех индексов, которые соответствуют численному значению признака, ставится 1. По сути, One-hot кодирование преобразует вектора категориальных признаков в матрицы бинарных признаков. Это позволяет придавать хоть какое-то осмысление числовым значениям категориальных признаков. В библиотеке Pandas это кодирование реализуется с помощью метода .get_dummies(). При этом можно преобразовывать полный датафрейм. One-hot кодирование применится только к столбцам с категориальными данными

```
DF_ohe = pd.get_dummies(DF.copy())
```

Стоит отметить, что использование One-hot кодирования значительно увеличивает размер матрицы данных. Поэтому для признаков с большим количеством уникальных значений рекомендуется предварительно уменьшить количество категорий, объединив редкие категории в одну, как было показано ранее.

Инженерия Признаков

Достаточно часто встречается, что с исходными признаками необходимо выполнить дополнительные преобразования, чтобы «помочь модели» выявить нужные закономерности. Как правило базовая инженерия признаков

сводится к «умной» комбинации исходных признаков. Рассмотрим несколько примеров для набора данных Cars Moldova.

Есть столбец данных Year (год выпуска автомобилей). Но с точки зрения предсказания стоимости автомобиля сам год как таковой имеет не очень большое значение. Большей смысл несет «возраст» автомобиля, т.е. количество лет, которое прошло от года выпуска до настоящего времени.

```
DF['Age'] = 2022 - DF.Year
```

Другим кажущимся логичным признаком является средний пробег в год, т.е. насколько интенсивно автомобиль использовался в среднем за год. Это чуть более полезная информация, чем просто общий пробег.

```
DF['km_year'] = DF.Distance/DF.Age
```

Инженерия новых признаков – достаточно творческий процесс, и иногда требует чуть более глубокого погружения в предметную область. При этом можно проверять различные гипотезы, например, создать категориальный признак, связанный с объемом двигателя: если объем двигателя больше, допустим, 3 литров то его стоит пометить отдельной категорией. Это связано с тем, что автомобили с большим объемом двигателя меньше востребованы из-за большего расхода и повышенным налогообложением. Это тоже должно сказываться на цене.

При добавлении признаков в набор данных, на основе имеющихся, важно отслеживать, а не добавляем ли мы в модель «то же самое что уже есть». Рекомендуется отслеживать коэффициент корреляции между признаков, и если он близок к единице, то какой-то из признаков рекомендуется удалить.

В библиотеке Pandas вычисление матрицы корреляции делается с использованием метода .corr(). Однако для лучшего восприятия информации добавим «раскраску» полученной результатов

```
cm = sns.color_palette("vlag", as_cmap=True)
```

```
DF.corr().style.background_gradient(cmap=cm, vmin = -1, vmax=1)
```

В результате вы должны получить раскрашенный датафрейм, похожий на Рис. 5-9

	Year	Distance	Engine_capacity(cm3)	Price(euro)	Age	km_year
Year	1.000000	-0.434034	-0.025707	0.551627	-1.000000	0.426025
Distance	-0.434034	1.000000	0.067378	-0.347236	0.434034	0.462777
Engine_capacity(cm3)	-0.025707	0.067378	1.000000	0.382831	0.025707	-0.010386
Price(euro)	0.551627	-0.347236	0.382831	1.000000	-0.551627	0.157024
Age	-1.000000	0.434034	0.025707	-0.551627	1.000000	-0.426025
km_year	0.426025	0.462777	-0.010386	0.157024	-0.426025	1.000000

Рис. 5-9 Матрица корреляция для набора данных Cars Moldova

Видно, что признаки Age и Year имеют 100 обратную корреляцию. Что не удивительно, ведь один признак — это константа минус второй признак. С другой стороны, корреляция между новым признаком km_year не превышает 0.5, что говорит о том, что этот признак может нести дополнительную информацию.

Ссылки на справочные материалы

1. Документация библиотеки Pandas для работы с данными https://pandas.pydata.org/
2. Документация библиотеки Seaborn для визуализации данных https://seaborn.pydata.org/

Примерные Вопросы для контроля

1. Опишите разные ситуации в которых вы будете использовать разные типы предварительной обработки данных (вопрос-дискуссия)
2. Как правильно использовать категориальные признаки в линейных моделях? (One-hot encoding)

6. **Регрессия**

Список тем, которые должны быть осуждены на лекции:

1. Регрессия
2. Метод Наименьших Квадратов
3. Градиентный Спуск
4. Регуляризация
5. Метрики Регрессии

Ключевые моменты по темам:

1. Задача регрессии. Обсуждение примеров задач из реальной жизни, которые можно свести к регрессии.
   Области применения задач регрессии (дискуссия)
2. Модель линейной регрессии
   Метод наименьших квадратов для поиска коэффициентов линейной регрессии
   Ограничение метода наименьших квадратов
3. Модель линейной регрессии
   Градиентный спуск для поиска коэффициентов линейной регрессии
4. Регуляризация
   Взаимосвязь величины коэффициентов линейной регрессии и высокой дисперсией
   L1 и L2 регуляризация
   SWOT-анализ линейной регрессии
5. Обсуждение метрик регрессии

Подведение Итогов Лекции

Текстовые материалы

Интересный исторический факт.

Два выдающихся математика заявили об авторстве и 200 лет спустя вопрос остается нерешенным. В 1805 году французский математик Адриан-Мари Лежандр опубликовал метод подгонки линии к набору точек. Он пытался предсказать местонахождение кометы (для справки в то время астронавигация была наукой, наиболее ценной в мировой торговле в то время, так же, как и «Искусственный Интеллект – новое электричество» в наше время).

Четыре года спустя 24-летний немецкий вундеркинд Карл Фридрих Гаусс настаивал на том, что он использовал тот же метод с 1795 года, но считал его слишком тривиальным, чтобы о нем писать. Утверждение Гаусса побудило Лежандра анонимно опубликовать приложение, в котором отмечалось, что «один очень знаменитый геометр без колебаний присвоил этот метод».

Дальнейшее развитие раскрыло широкий потенциал алгоритма. В 1922 году английские статистики Рональд Фишер и Карл Пирсон показали, как линейная регрессия вписывается в общую статистическую структуру корреляции и распределения, что сделало ее полезной во всех науках. И почти столетие спустя появление компьютеров предоставило данные и вычислительную мощность, позволяющую извлечь из них гораздо больше пользы.

Стоит также отметить, что наиболее распространенным типом нейронов в нейронной сети является модель линейной регрессии, за которой следует нелинейная функция активации, что делает линейную регрессию фундаментальным строительным блоком глубокого обучения.

Генерируемые Данные

Перед тем, как использовать алгоритм на реальных данных в учебных целях бывает полезно тестировать его на данных, которые мы можем менять самостоятельно. Чтобы «покрутить» различные ситуации и смотреть, как это влияет на результат.

Реализуем с помощью функций true_fun создание одномерных данных, которые могут быть некоторыми функциями от входных данных X.

```
def true_fun(x, a=np.pi, b = 0, f=np.sin):
    x = np.atleast_1d(x)[:]
    a = np.atleast_1d(a)

    if f is None: f = lambda x:x
    x = np.sum([ai*np.power(x, i+1) for i,ai in enumerate(a)],axis=0)

    return f(x+ b)
```

В этой функции x - входные данные, a - коэффициент, на который данные умножаются, b – константная добавка, f – функция, которую мы применяем к входным данным. Стоит отметить, что для полиномиальных зависимостей достаточно использовать в качестве a список коэффициентов. Созданные данные могут быть как линейными, f=None, так и гармоническими (np.sin, np.cos) или экспоненциальными np.exp.

В учебных целях (чтобы не иметь всегда 100% точности) мы также добавим шумы к данным с помощью функции noises. Аргументами этой функции являются shape (размер массива) и noise_power (величина шума).

```
def noises(shape , noise_power):
    return np.random.randn(*shape) *noise_power
```

Соберем это все в рамках единой функции dataset. Новыми аргументы в этой функции являются N – количество точек, которые мы хотим сгенерировать,

x_max – максимальное значение входных данных и булева переменная random_x. Если random_x = True, тогда будут случайным образом сгенерированы N точек, иначе данные будут распределены равномерно в диапазоне от 0 до x_max.

```
def dataset(a, b, f = None,  N = 250, x_max =1, noise_power = 0, random_x = True
,  seed = 42):
    np.random.seed(seed)

    if random_x:
        x = np.sort(np.random.rand(N))*x_max
    else:
        x = np.linspace(0,x_max,N)

    y_true = np.array([])

    for f_ in np.append([], f):
        y_true=np.append(y_true, true_fun(x, a, b, f_))

    y_true = y_true.reshape(-1,N).T
    y = y_true + noises(y_true.shape , noise_power)

    return y, y_true, np.atleast_2d(x).T
```

Выходными параметрами этой функции являются y_true (истинный вариант зависимости), y (зашумленный вариант), x (входные данные).

Приведем примеры команд для генераций данных и визуализацию данных:

- линейная зависимость $y = 2x + 2,\ x \in [0, ..., 1]$ (Рис. 6-1)

y, y_true, x = dataset(a = 2, b = 2, f = None, N = 100, x_max =1, noise_power = 0.1, seed = 42)

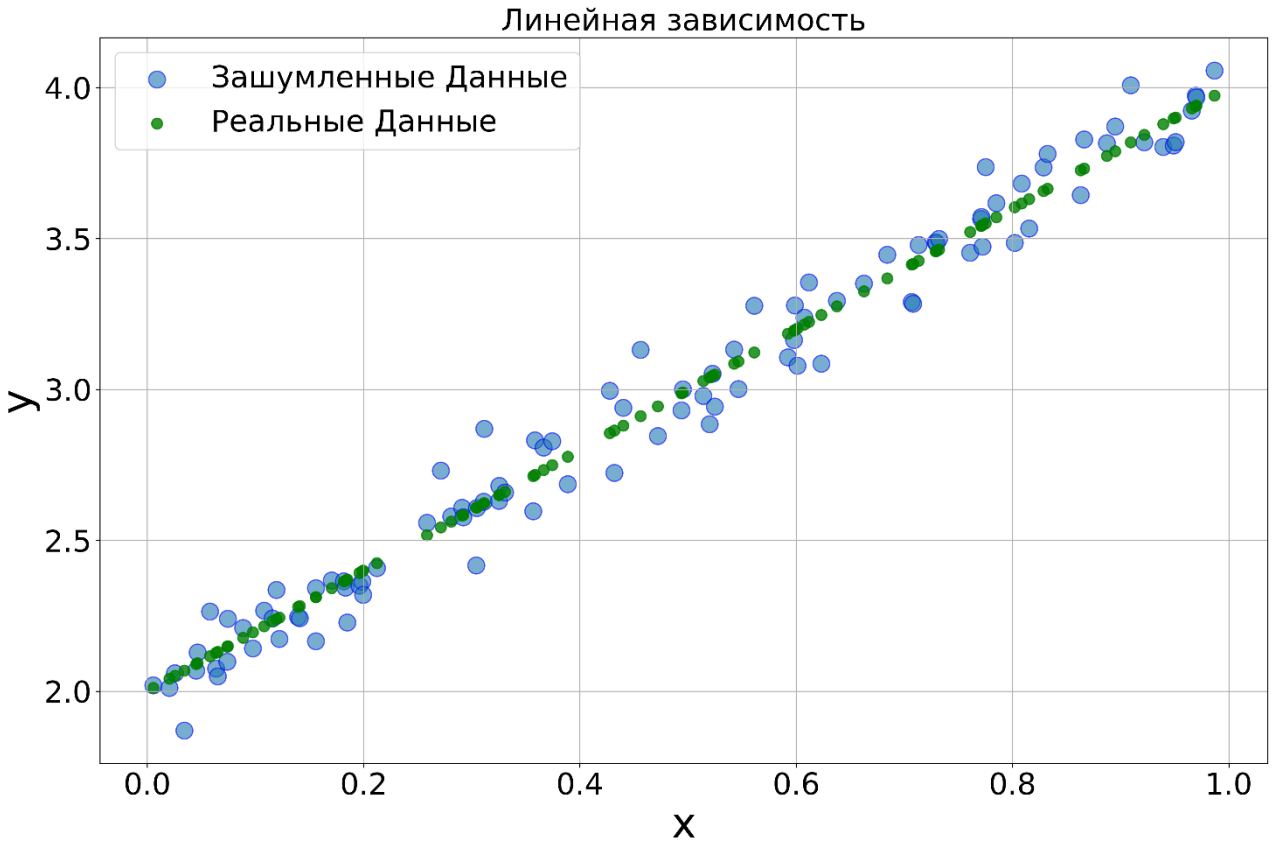


Рис. 6-1 Сгенерированные данные с линейной зависимостью

- полиномиальная зависимость

$$y = -2x^3 + 2x^2 - x - 1,\ \ x \in [0, ..., 1.25]$$ (Рис. 6-2)

y, y_true, x = dataset(a = [1,2,-2], b = -1, f = None, N = 250, x_max =1.25, noise_power = 0.05, seed = 42)

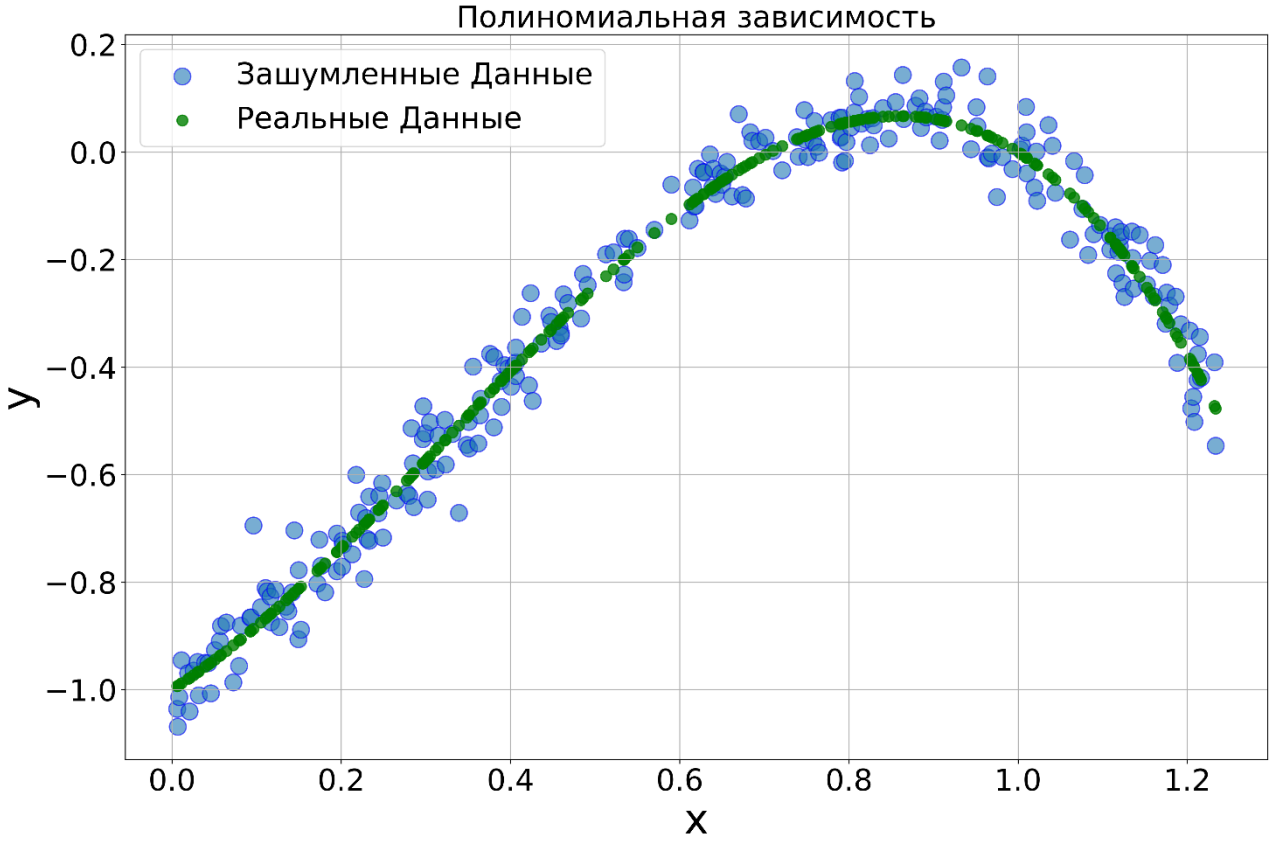


Рис. 6-2 Сгенерированные данные с полиноминальной зависимостью

- гармоническая зависимость $y = \cos cos\,(3\pi x + \pi)$, $x \in [0, ..., \frac{\pi}{4}]$

y, y_true, x = dataset(a = 3*np.pi,b = 0, f = np.cos, N = 25, x_max =np.pi/4, noise _power = 0.1, seed = 42)

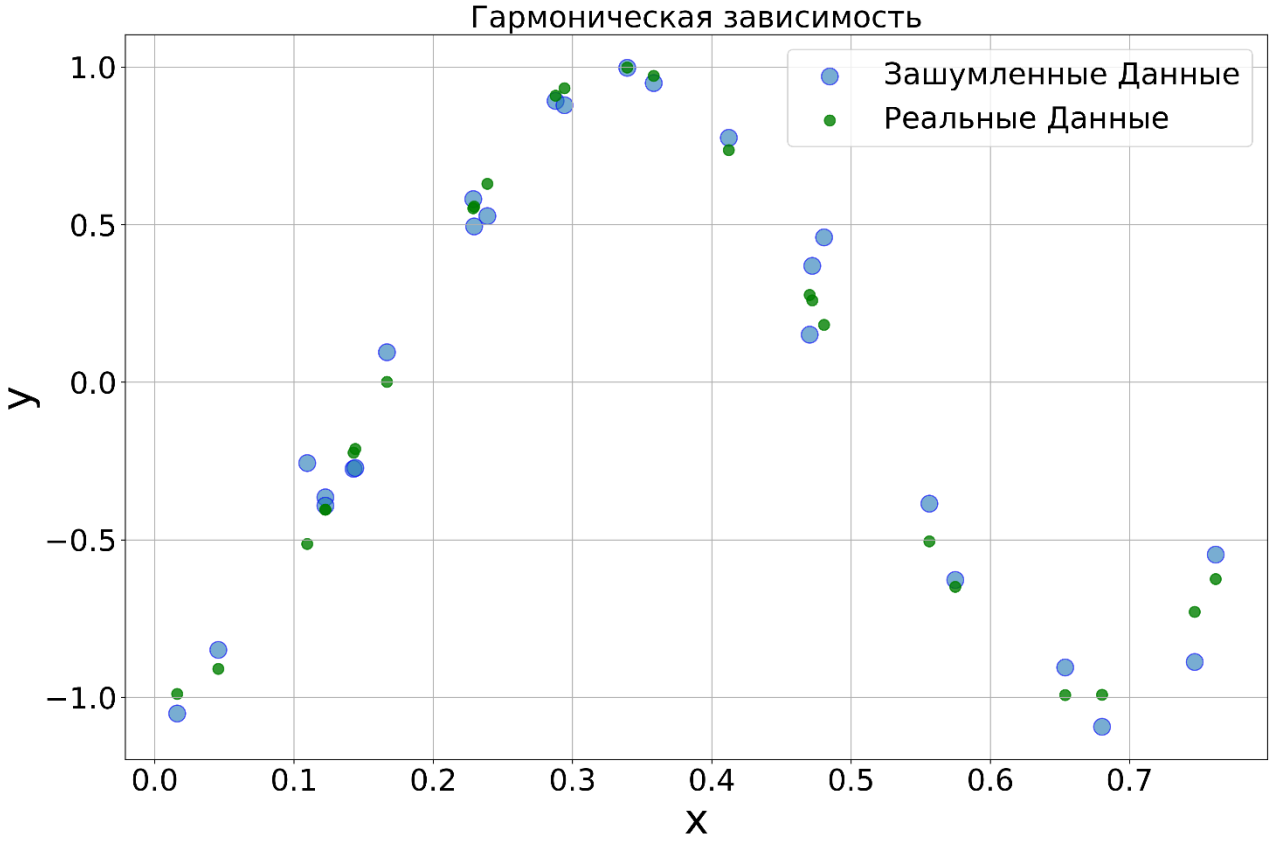


Рис. 6-3 Сгенерированные данные с гармонической зависимостью

Модель линейной регрессии

Для начала попытаемся построить самую простую зависимость – линейную, которая представлена на (Рис. 6-1). Проведем линейную регрессию. То есть оценим коэффициенты аппроксимации зашумленных данных функцией вида

$$\hat{y} = \hat{a}x + \hat{b},$$

где $\hat{y}_i$ — это наши оценки целевых значений y_i; $\hat{a}$ и $\hat{b}$ - коэффициенты модели.

В самом простом случае (для нашего набора данных) мы могли бы найти коэффициенты $\hat{a}$ и $\hat{b}$ аналитически. Если бы не было шумов (была бы чистая линия), коэффициенты можно найти классическим решением линейной системы из 2 независимых уравнений. В случае шумов 2 уравнений может

быть недостаточно. Оценки коэффициентов по каждым двум уравнениям могут различаться. Поэтому желательно иметь переопределенную систему (когда число уравнений больше числа переменных). Такую систему можно решать по-разному.

Известно (теорема Гаусса-Маркова), что если шум в системе имеет нормальное распределение, то оптимальным будет решение методом наименьших квадратов. Для поиска такого решения предположим, что мы производим такие оценки коэффициентов $\hat{a}$ и $\hat{b}$, которые дают результат $\hat{y}$ с ошибкой ε. Ошибку определим как $\varepsilon_i = y_i - \hat{y}_i$. Теперь наша задача минимизация ошибки ε для каждого значения x. В случае метода наименьших квадратов задача сводиться к минимизации одного значения среднего квадратов ошибки. Для линейной регрессии такую задачу можно решить аналитически. Однако, как можно будет увидеть ниже такое решение не всегда рационально.

В более общем случае, допустим, что мы не знаем, как лучше всего аппроксимировать зависимость, какую функцию использовать. Такая ситуация характерна для зависимостей, более сложных чем линейная. В таких случаях можно предположить, что указанной выше модели недостаточно. В более общем случае можно ввести модель целевой переменной как

$$y_i = \sum_{j=0}^{m} x_{ij} w_j + \eta_i \text{ или так } y_i = x_{i0} \cdot w_0 + x_{i1} \cdot w_1 + x_{i2} \cdot w_2 + \ldots + x_{im} \cdot w_m + \eta_i \text{ или}$$

где y_i - целевой показаль предсказания для i записи в наборе данных;

$X_i = \{x_{ij}\}_{j=1}^{m}$ - набор входных параметров для i результата; $W = \{w_j\}_{j=1}^{m}$ - набор весовых параметров, которые мы должны подобрать в модели; η_i - некоторый набор случайных (не объясняемых нашей моделью, остаточных) значений, мы будем считать их случайных шумом.

Стоит отметить, что в формуле суммирование происходит по индексу j от 0. Это связано с тем, что как правило в модель, добавляют «нулевой» столбец, которые для всех i равен 1. Это позволяет оценить независимый коэффициент w_0, который иногда называют смещением (bias). Однако ее не обязательно добавлять в модель.

Данная модель соответствует как линейной регрессии одной или нескольких переменных, так и полиномиальной или любой другой, где признаки (x_{ij}) можно считать независимыми составляющими. В качестве примера регрессионная модель одной переменной будет иметь вид

$$\hat{y}_i = \sum_{j=0}^{1} x_{ij} w_j \text{ или так } \hat{y}_i = 1 \cdot w_0 + x_{i1} \cdot w_1 \text{ или так } \hat{y}_i = X_i W^T,$$

где $\hat{y}_i$ - результат предсказания для i записи в наборе данных.

Введем функцию расчета (предсказания) значений predict. Здесь мы задали возможность добавлять или нет в модель независимы коэффициент с помощью булевой переменной add_bias. Для учета смещения мы будем добавлять единичный столбец к входным данным.

```
def predict( X, weights, add_bias = True):
    if add_bias:
        X_full = np.column_stack((np.ones(X.shape[0]),X))
    else:
        X_full = X
    return  np.dot(X_full, weights)
```

Теперь рассмотрим решение для обозначенной модели одной переменной. Введем функцию потерь регрессии loss_func как квадрат разности между целевыми значениями и их предсказаниями

$$L(\hat{y}_i, y_i) = L_i = (\hat{y}_i - y_i)^2 = \left(\sum_{j=0}^{2} x_{ij} w_j - y_i\right)^2 = \left(X_i W^T - y_i\right)^2,$$

где L_i - функция потерь для результата (предсказания) с номером i.

В виде функций Python это реализуется достаточно просто

```
def loss_func(yhat, y):
    return np.square(yhat - y)
```

Отметим, что для нашего случая одной переменной (в выше приведенных обозначения) решение могло бы быть найдено как

$$L = \sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2 = \sum_{i=0}^{N-1} (y_i - (a \cdot x_i + b))^2 \to 0$$

тогда минимум L будет соответствовать нулям ее производных по a и b:

$$\{\frac{\partial L}{\partial a} = 2 \sum_{i=0}^{N-1} (y_i - (a \cdot x_i + b)) x_i = 0 \; \frac{\partial L}{\partial b} = 2 \sum_{i=0}^{N-1} (y_i - (a \cdot x_i + b)) = 0$$

Отсюда решение систему уравнений выглядит как

$$\{a = \frac{N\sum xy - \sum x \sum y}{N\sum x^2 - (\sum x)^2} \; b = \frac{\sum y - a\sum x}{N}$$

В более общем случае мы могли бы записать уравнение как

$$L = \sum_{i=0}^{N-1} (y_i - X_i W^T)^2 = 0 \text{ или } (y - XW^T)^2 = 0$$

тогда

$$\frac{\partial L}{\partial W} = 2(y - XW^T)X = 0$$

или

$$W = (X^T X)^{-1} X^T y = X^+ y$$

Однако, если массивы X и y достаточно большие, то такое решение оказывается весьма вычислительно сложным. Для больших массивов данных чаще используют численные методы оптимизации. Среди таких методов наиболее зарекомендовал себя метод градиентного спуска.

Метод градиентного спуска позволяет итерационно решить задачу оптимизации. В дискурсе данного пособия мы будем называть такой процесс оптимизации **обучением**. Каждую итерацию принято называть **эпоха**. Ниже будет показ принцип обучения методом градиентного спуска и его реализация.

Важно отметить, что особенностью итеративных методов обучения является потенциальная ситуация переобучения/недообучения в ходе оптимизации. Проще всего это представить, как ситуацию, в которой мы ошибемся в выбранных значениях коэффициентов. Такое явление может происходить если, например, наша модель будет воспринимать все шумы, помехи и искажения входных данных как важные для точного ответа. Другими словами, для данных, участвующих в обучении (обучающая выборка), наша ошибка будет стремиться к нулю. Однако, для данных, отличных от обучающей выборке точность будет не высокой.

Чтобы не допустить этого на каждом шаге обучения мы будем проверять полученные коэффициенты модели. Для такой проверки мы будем использовать т. н. *валидационную* выборку. Как правило *валидационная* и *тренировочная* выборки выбирается из одних и тех же данных. В некоторых случаях, кроме данных выборок также может быть и третья выборка,

независимая от двух указанных. Такая выборка будет необходима для проверки итоговой точности модели. Итоговую проверочную выборку можно назвать *тестовая* выборка. По существу, тестовая выборка характеризует т. н. обобщающую способность, то есть разность между точностью на *тренировочных* данных и данных, в которых модель должна работать. Разность значений точности должна быть как можно меньше.

Для того, чтобы выделить из входных данных тренировочную и тестовую выборку запишем следующую функцию train_test_split. Функция будет иметь входные аргументы:

- x,y - входные данные и целевые значения;
- train_size - размер тренировочной части;
- test_size =None - размер тестовой части;
- random_state=None - состояние генератора случайных чисел;
- shuffle=True - необходимость перемешивания данных.

```
def train_test_split(x,y, train_size=None, test_size=None, random_state=42, shuffle=True
,):
    if random_state: np.random.seed(random_state)

    size = y.shape[0]
    idxs = np.arange(size)
    if shuffle: np.random.shuffle(idxs)

    if test_size and train_size is None:
        if (test_size<= 1):  train_size = 1 - test_size
        else:  train_size = size - test_size
        test_size = None

    if train_size is None or train_size > size: train_size = size

    if (train_size<= 1): train_size *= size
```

```
    if test_size is not None:
        if test_size<= 1: test_size *= size
        if test_size>size: test_size = size-train_size
    else: test_size = 0


    x_train, y_train = x[idxs[:int(train_size)]], y[idxs[:int(train_size)]]
    x_val,   y_val   = x[idxs[int(train_size):size - int(test_size)]], y[idxs[int(train_size):size
- int(test_size)]]


    if test_size > 0:
        x_test, y_test = x[idxs[size - int(test_size):]], y[idxs[size - int(test_size):]]
        return x_train, y_train.squeeze(), x_val, y_val.squeeze(), x_test, y_test.squeeze()
    return x_train, y_train.squeeze(), x_val, y_val.squeeze()
```

Если есть необходимость разбить и на тренировочную, и на валидационную и на тестовую тогда нужно указать и размер тестовой выборки и тренировочной. Размер в валидационную выборку войдут оставшиеся данные

```
x_train, y_train, x_val, y_val, x_test, y_test = train_test_split(x, y, train_size = 0.5,
test_size=0.3, )
```

В этом случае размер тренировочной выборки составит 50 точек, валидационной – 20 точек, тестовая – 30 точек.

Чтобы разбить исходные данные только на тренировочную и тестовую выборку достаточно указать только размер тестовой выборки

```
x_train, y_train,  x_test, y_test = train_test_split(x, y,  test_size=0.3, )
```

Тогда исходная выборка из 100 точек разобьется на тренировочную выборку, в которой 70 точек и тестовую в которой 30 точек.

Далее функции будут тестироваться для фиксированного random_state = 42, и разбиения исходных данных только на тренировочную и тестовую (test_size=0.3).

Перед началом процедуры обучения модели запишем функцию инициализации весовых параметров (коэффициентов модели) init_weights. Данная функция будет создавать случайный массив весовых параметров с нормальным распределением, имеющим среднее равное 0 и разброс значений $1/\sqrt{W\_shape}$.

Также мы будем иметь возможность создавать набор весов с учетом смещения. То есть, если add_bias = True, то размер выходного массива на 1 больше, чем размер признаков входных данных (в нашем случае входные данные имеет 1 признак, а число параметров будет 2: коэффициент смещение и при признаке). Значения смещения проинициализируем нулями.

```
def init_weights(W_shape, add_bias = True, random_state = 42):
    W_shape = np.atleast_1d(W_shape)
    if random_state:
        np.random.seed(random_state)
    weights = np.random.randn(*list(W_shape))/np.sqrt(np.sum(W_shape))
    if add_bias:
        weights = np.column_stack((np.zeros(W.shape[-1]), weights ))
    return weights.squeeze()
```

Протестируем функции: инициализируем веса и протестируем модель для первой строчки данных

```
weights = init_weights(x.shape[1])

yhat = predict( x_train[0],weights)

loss = loss_func(yhat, y[0])
```

Если вы все запустили правильно и использовали везде random_state = 42, то вы должны получить вектор весов $w_0 = 0.$, $w_1 = 0.49671415$, предсказание модели $\hat{y}_0 = 0.4015424$, реальном значении целевой переменной $y_0 = 2.01974894$ и значение функции потерь $loss\left(\hat{y}_0\right) = 2.6185924$. А что вы хотели, веса сгенерированы случайным образом, не удивительно что предсказания модели настолько разняться с реальными значениями. Давайте попытаемся оптимизировать веса.

Оптимизацию мы будем проводить методом градиентного спуска. По сути этот метод сводится к последовательному (итерационному) пересчету значений весовых параметров обратно значениям градиента ошибки (то есть в направлении, обратном направлению роста ошибки).

$$\frac{\partial L_i}{\partial w_j} = 2(\hat{y}_i - y_i)x_{ij},$$

где $\frac{\partial L_i}{\partial w_j}$ - частная производная функции L_i по параметру w_j

тогда по набору всех переменных мы получим производную вида:

$$\nabla_W L_i = 2\{\left(\hat{y}_i - y_i\right)x_{i0}, \left(\hat{y}_i - y_i\right)x_{i1}, \left(\hat{y}_i - y_i\right)x_{i2}\} = 2(\hat{y}_i - y_i)\odot X_i^T,$$

где $\nabla_W L_i$ - градиент, то есть набор частных производных функции L_i по набору $\{w_j\}$. $\odot$ - операция поэлементного умножения (умножение Адамара).

Реализуем соответствующую функцию на Python с учетом возможности добавления смещения.

```
def grad_loss(y_hat, y, X, add_bias = True):
    if add_bias:
        X_full = np.column_stack((np.ones(X.shape[0]),X))
    else:
        X_full = X
    return 2*np.dot(X_full.T, (y_hat - y)) / y.size
```

Оценим градиент весов для первой точки

grad = grad_loss(yhat, y[0], x[0])

Вы должны получить следующие значения [-3.23641307 , -0.01787185]

Обозначим номер итерации как t, тогда выражение для обновления весовых параметров можно записать как:

$$W^t = W^{t-1} - \eta\nabla_W L(\hat{y}_i, y_i) = W^{t-1} - 2\eta(\hat{y}_i - y_i)\odot X_i^T$$

где η - коэффициент, с которым изменяются значения весовых параметров - т.н. скорость обучения (learning rate).

Реализуем это в виде функции update_weights

```
def update_weights(grad, weights, learning_rate):
    return weights - learning_rate*grad
```

Теперь проведем обновление весовых параметров, и оценим

weights = update_weights(grad, weights, 0.1)

После 1 итерации обновления весов должны получится следующие значения $w_0 = 0.32364131$, $w_1 = 0.49850134$, предсказание модели для нулевой точки $\hat{y}_0 = 0.72662847$, а функция потерь $loss\left(\hat{y}_0\right) = 1.67216056$. Уже лучше, но надо повторить операцию оценки градиента весов и последующего обновления несколько раз.

Создадим процедуру итерационного обучения. Процедура будет повторять процесс пересчета весов методом градиентного спуска заданное число раз (epochs). Функция будет требовать на вход X - набор входных значений в формате: число записей × признаки в записи; y - набор целевых переменных; weights - начальные значения весовых параметров; learning_rate - скорость обучения; epochs- число эпох обучения.

Функция дает на выходе: weights- набор обученных весовых параметров; cost – значение функционала потерь на каждой эпохе обучения. Также отметим, что на практике мы можем обновлять весовые параметры не для каждого отдельного значения i, а для целого набора такхи значений, тогда более верное выражение будет выглядеть как

$$W^t = W^{t-1} - \eta \frac{1}{N} \sum_{i=0}^{N-1} \nabla_W L(\hat{y}_i, y_i),$$

где N - объем выборки.

```
def fit(X, y, learning_rate, weights = None,  epochs=30):

    if weights is None: weights = init_weights(X.shape[1])
    cost    = np.zeros(epochs)

    for i in range(epochs):
        yhat    = predict(X,weights)
```

```
        grad    = grad_loss(yhat,  y, X)
        weights = update_weights(grad, weights, learning_rate)
        cost[i] = loss_func(yhat,  y).mean()

    return weights, cost
```

Протестируем обучение на 10 эпохах и при значении скорости обучения 0.1

```
weights, cost = fit(x_train, y_train, learning_rate=0.1, epochs=25)
```

Можно визуализировать полученные значения функционала потерь для каждой эпохи обучения (Рис. 6-4)

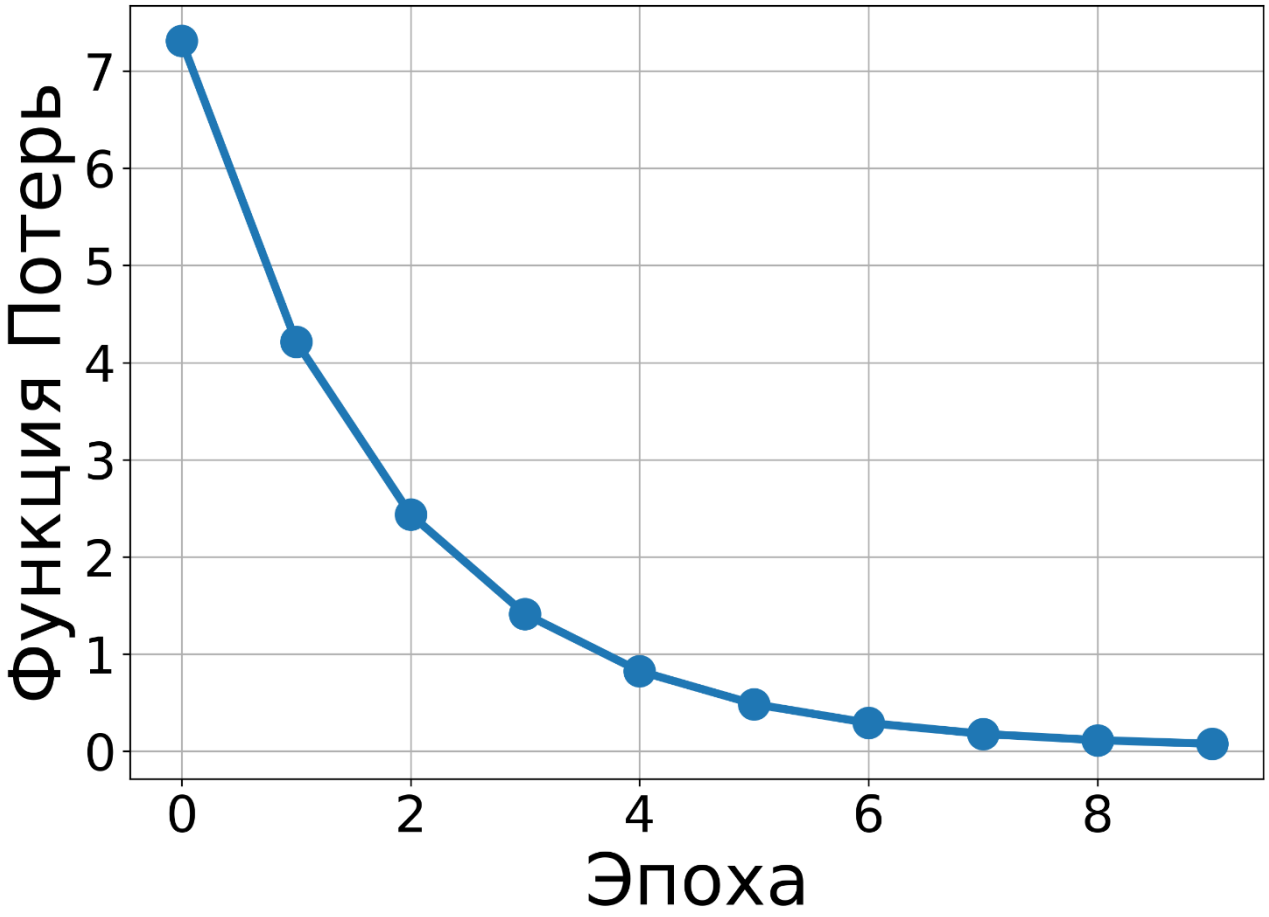


Рис. 6-4 Зависимость функционала потерь от эпохи для модели, обученной с помощью функции fit

Как видно за 10 эпох обучения функционал потерь значительно уменьшился. Визуализируем предсказания модели (Рис. 6-5). При этом получены следующие веса $w_0 = 2.04548417$, $w_1 = 1.55145866$, что достаточно близко к реальным значениям.

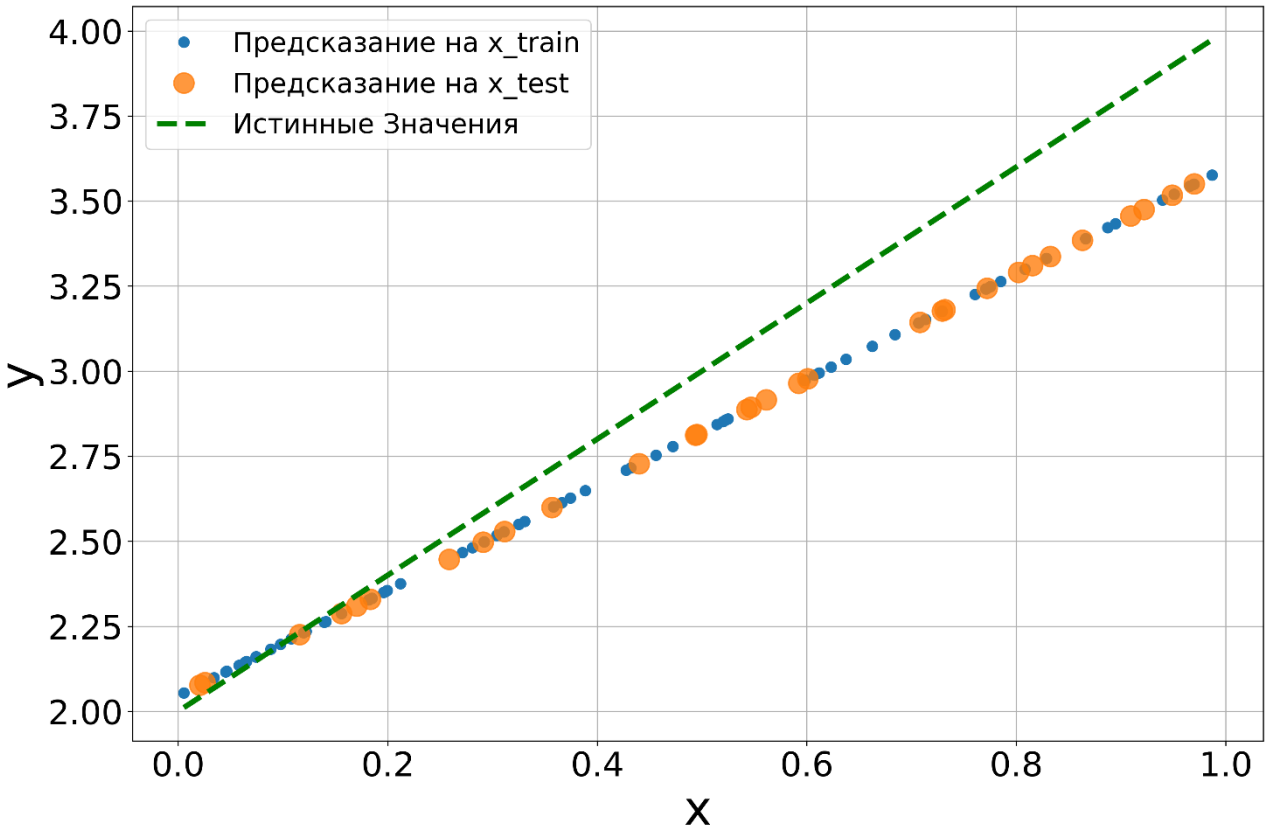


Рис. 6-5 Полученная модель с помощью функции fit

Выглядит «похоже на правду», но давайте количественно оценим модель. Для этого мы можем воспользоваться метрикой Коэффициент детерминации (Coefficient of Determination) R^2. Метрика соответствует относительной среднеквадратичной ошибке, она может быть рассчитана как:

$$R^2 = 1 - \frac{\sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{N-1} (y_i - \bar{y})^2},$$

где $\bar{y} = \frac{1}{N} \sum_{i=0}^{N-1} y_i$ - среднее значение.

```
def r2_score(yhat, y):
    return 1-(np.square(y-yhat)).sum(axis=0)/(np.square(y-np.mean(y, axis=0))).sum
(axis=0)
```

Отметим, важное обстоятельство. Для расчета градиента мы использовали функцию потерь. Однако, для оценки качества модели мы пользуемся другой функцией метрикой. Дело в том, что значения функции потерь, сколь небольшими бы они не были, очень сложно интерпретировать. Более того, можно ожидать, что для другого метода оптимизации значения могли быть и другими. Таким образом значения функции потерь не подходят для оценки качества модели. Качество работы модели как правило определяется по некоторым метрикам. Такие метрики должны быть интерпретируемыми и едиными для всех сравниваемых оценщиков. В нашем случае метрика соответствует среднему квадрату ошибки, отнесенному к дисперсии целевых значений. Тогда результат работы метрики мы можем интерпретировать как средняя ошибка предсказания для нашей модели. Чем ниже эта ошибка (для разных моделей), тем лучше.

Помимо коэффициента детерминации для оценки моделей регрессии могут использоваться следующие метрики:

- Mean Square Error Среднеквадратичная ошибка

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Root Mean Square Error Средняя квадратическая ошибка

$$RMSE = \sqrt{MSE}$$

- Mean Average Error Средняя абсолютная ошибка

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- Mean Absolute Percentage Error Средняя абсолютная процентная ошибка

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

- Mean Squared Logarithmic Error Среднеквадратическая логарифмическая ошибка

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}\{ln(1 + y_i) - ln(1 + \hat{y}_i)\}^2$$

- Максимальная ошибка

$$MaxError = max(|y_i - \hat{y}_i|)$$

Применим нашу функцию r2_score. Для тренировочных данных должно получиться значение 0.8570755941810686, а для тестовых - 0.818294732486739.

Очевидным способом увеличения качества модели является увеличение количества эпох. На практике как правило, на каждой эпохе рассматривается, не вся выборка, а только некоторая ее часть - т. н. батч (мини пакет). Это позволяет быстрее рассчитывать градиент весов и чаще обновлять веса. Запишем функцию для генерации батчей заданного размера batch_size

```
def load_batch(X,y, batch_size = 100):
    idxs = np.arange(y.size)
    np.random.shuffle(idxs)

    for i_batch in range(0,y.size,batch_size):
        idx_batch = idxs[i_batch:i_batch+batch_size]
        x_batch   = np.take(X, idx_batch,axis=0)
        y_batch   = np.take(y, idx_batch)
```

```
        yield  x_batch, y_batch
```

Обновим нашу функцию fit с учетом разбиения данных на батчи.

```
def fit_SGD(X, y, lerning_rate, weights = None,  epochs=30, batch_size = 100, random_state = 4
2):

    if random_state: np.random.seed(random_state)

    if weights is None: weights = init_weights(X.shape[1])
    if batch_size is None or batch_size>y.size : batch_size = y.size
    n_batches = y.size//batch_size

    cost    = np.zeros(epochs)

    for i in range(epochs):
        loss = 0
        for cnt,(x_batch, y_batch) in enumerate(load_batch(X,y, batch_size)):

            yhat    = predict(x_batch, weights)
            grad    = grad_loss(yhat,  y_batch, x_batch)
            weights = update_weights(grad, weights, lerning_rate)
            loss   += loss_func(yhat,  y_batch).mean()

            if cnt>= n_batches:
                break
        cost[i] = loss/n_batches

    return weights, cost
```

Таким образом на каждой эпохе обучения веса обновляются не 1 раз, а столько раз на сколько батчей можно разделить исходную выборку. В

предельном случае реализуется стохастический градиентный спуск, при котором обновление происходит при выборе 1 случайно выбранного элемента.

Визуализируем зависимость функционала потерь от эпох при обучении модели линейной регрессии с использованием функции fit_SGD (Рис. 6-6), а также визуализируем предсказания модели fit_SGD на тренировочных и тестовых данных (Рис. 6-7).

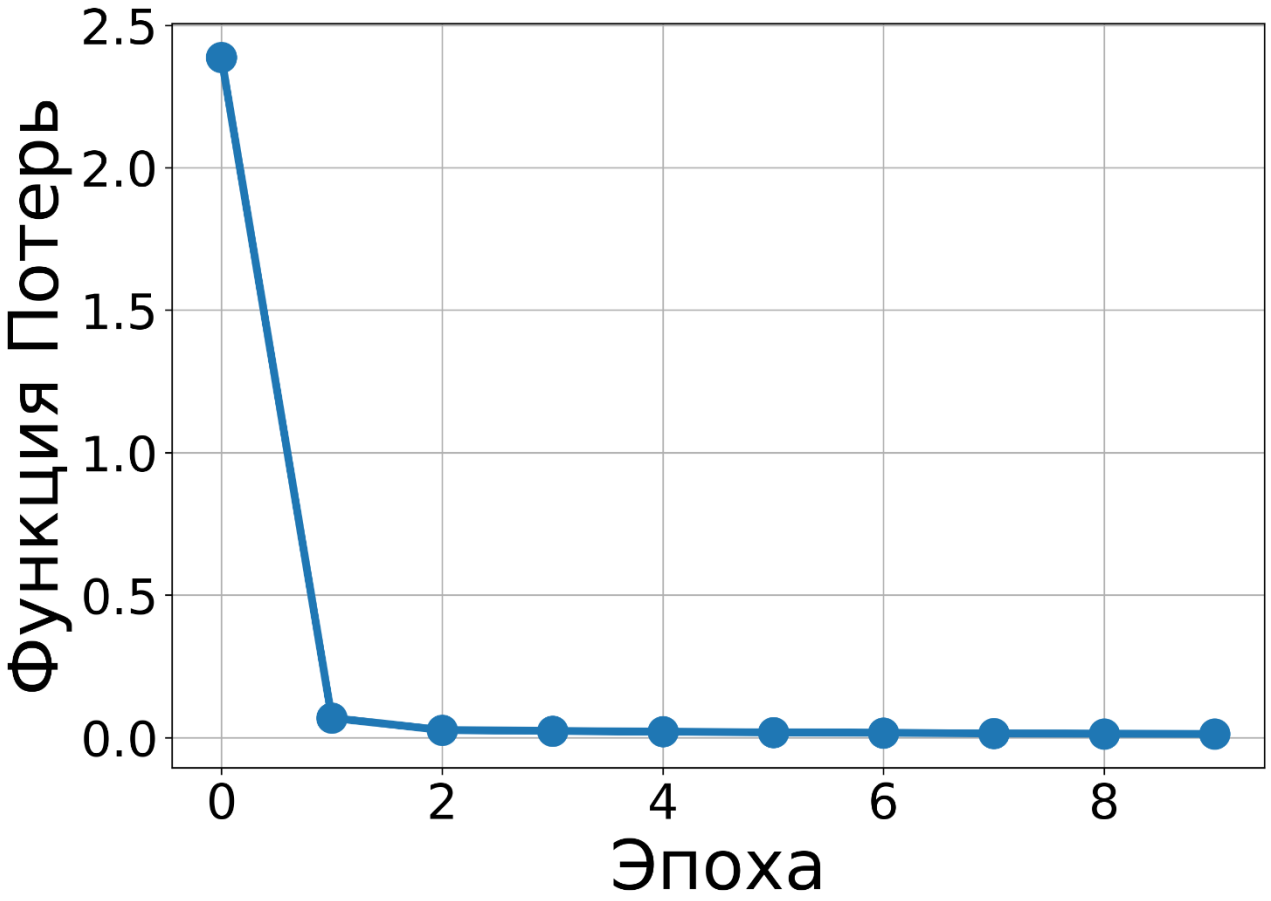


Рис. 6-6 Зависимость функционала потерь от эпохи для модели, обученной с помощью функции fit_SGD

При этом получены следующие веса $w_0 = 2.08073489$, $w_1 = 1.85005883$, что еще ближе к реальным значениям.

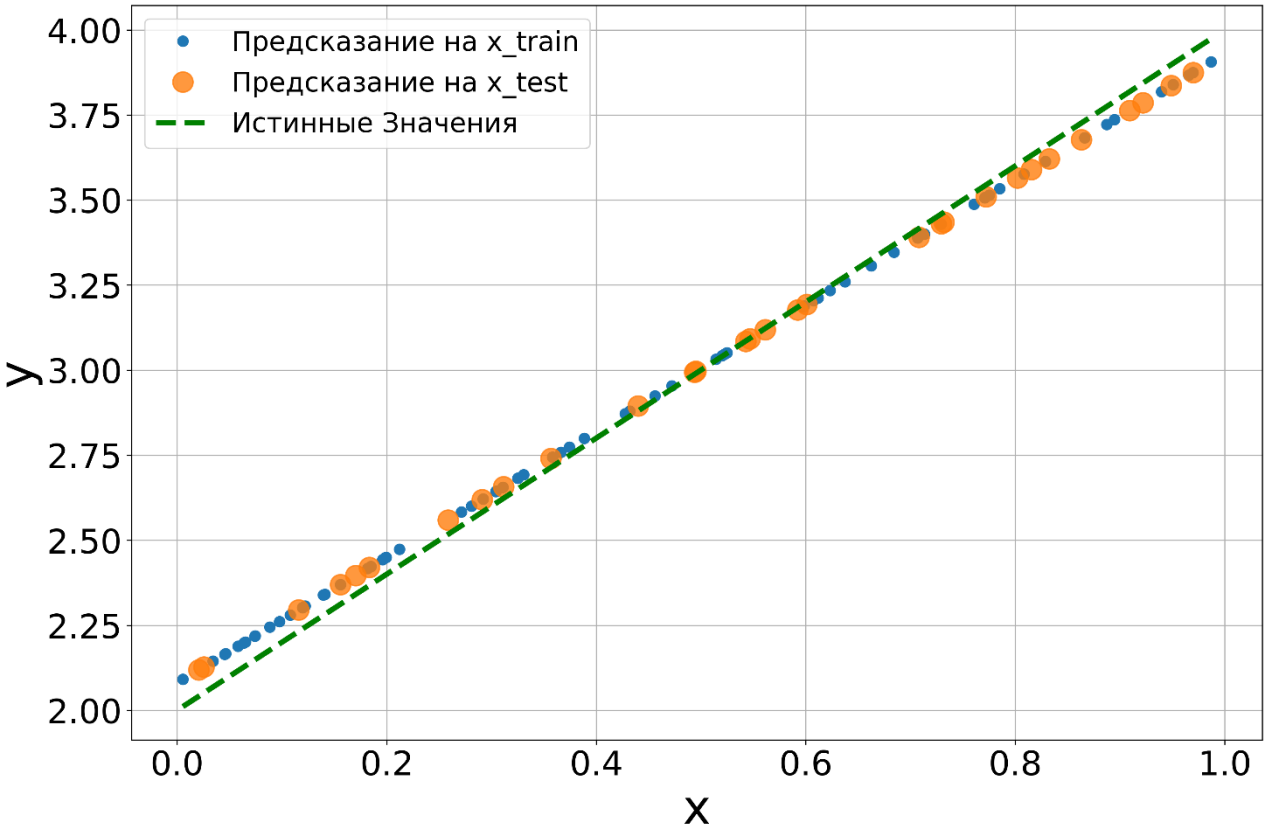


Рис. 6-7 Полученная модель с помощью функции fit_SGD

Для тренировочных данных должно получиться значение 0.9685919335230442, а для тестовых - 0.9715819769095048.

Мы получили достаточно интересный результат. Количество эпох было одинаковое, в обоих случаях мы использовали одинаковое число данных. Однако если мы предварительно разбиваем выборку, и обновляем веса на каждой части отдельно то результат обучения модели лучше. Такую идею можно применить и в реальной жизни – не обязательно решать большую задачу целиком, можно разбить ее на несколько подзадач.

Реализация описанного нами алгоритма не с помощью функций, как описано в этой главе, а с помощью единого класса LinearRegression. В основном методы, реализованные в этом классе, совпадают с описанными ранее функциями. Подход ООП (объектно-ориентированного программирования) позволит легче модифицировать алгоритм в последующих подглавах и главах.

Полиномиальная регрессия

Теперь попробуем реализовать применить разработанный нами алгоритм для полиномиальной зависимости (Рис. 6-2). Сколько бы вы ни добавляли эпох в обучения наиболее вероятно, что предсказания вашей модели будет выглядеть следующим образом (Рис. 6-8).

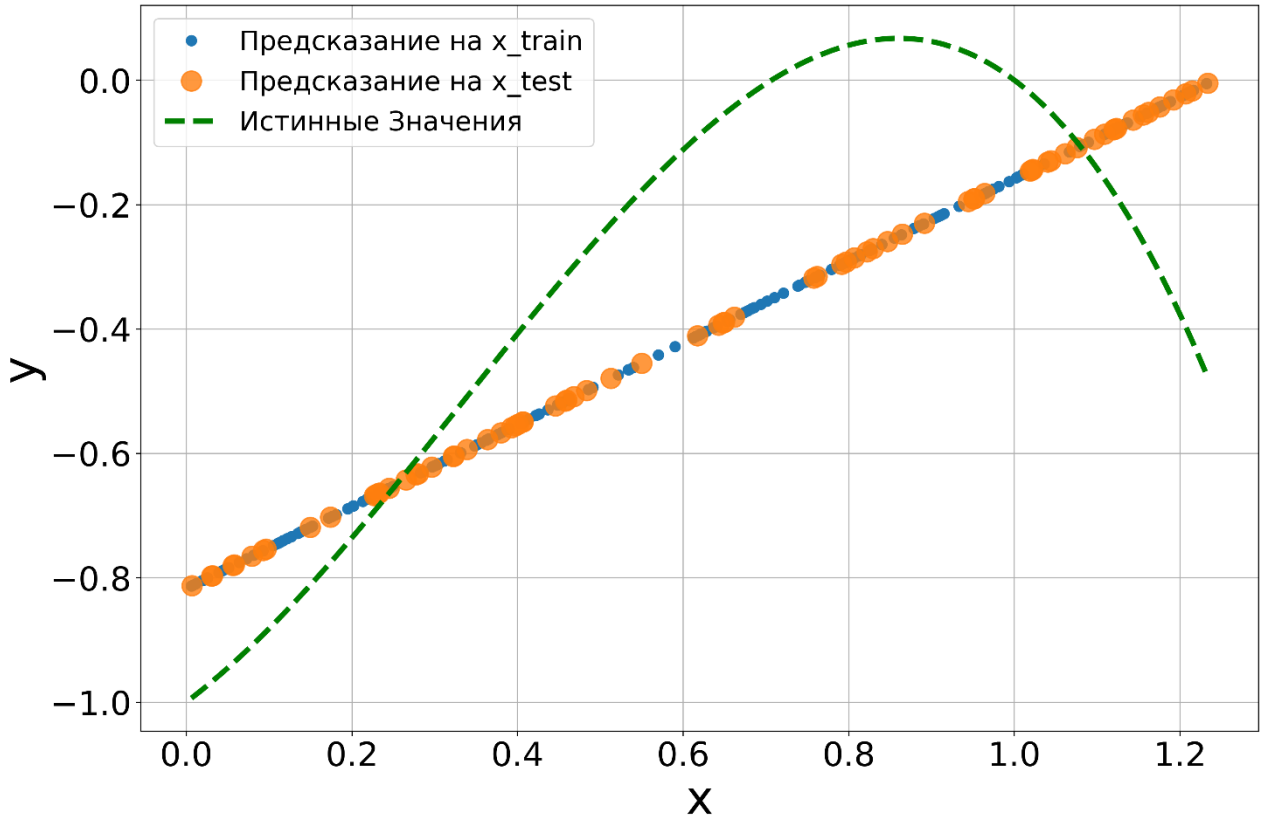


Рис. 6-8 Полученная линейная модель для полиномиальной зависимости

Видно, что линейная модель «слишком проста», для такого распределения истинных значений целевой переменной (высокое смещение). Давайте попробуем усложнить модель. И поскольку у нас не откуда сгенерировать дополнительные новые признаки будем использовать имеющиеся данные. Как известно члены полиномов различных целых степеней можно считать независимыми признаками. Поэтому полиномиальная регрессия может быть представлена как многопеременная линейная регрессия.

Для реализации давайте запишем функцию to_polynom создающую полином из входных данных X. Выход функции будет массив, имеющий число столбцов, равное степени искомого полинома.

```
def to_polynom(X, order = 2):
    order_range = range(order, order+1,1)
    out = np.copy(X)
    for i in order_range:
        out = np.hstack([out, np.power(X,i)])
    return out
```

Теперь если предварительно сгенерировать полиномиальные признаки и уже потом применить к ним алгоритм линейной регрессии можно получить гораздо лучший результат (Рис. 6-9).

```
x_ = to_polynom(x, order = 5)
x_train, y_train,  x_test, y_test = train_test_split(x_, y,  test_size=0.3, )
regr = LinearRegression(learning_rate=0.1,epochs=100,batch_size=10, n_batches=None)
regr.fit(x_train, y_train)
```

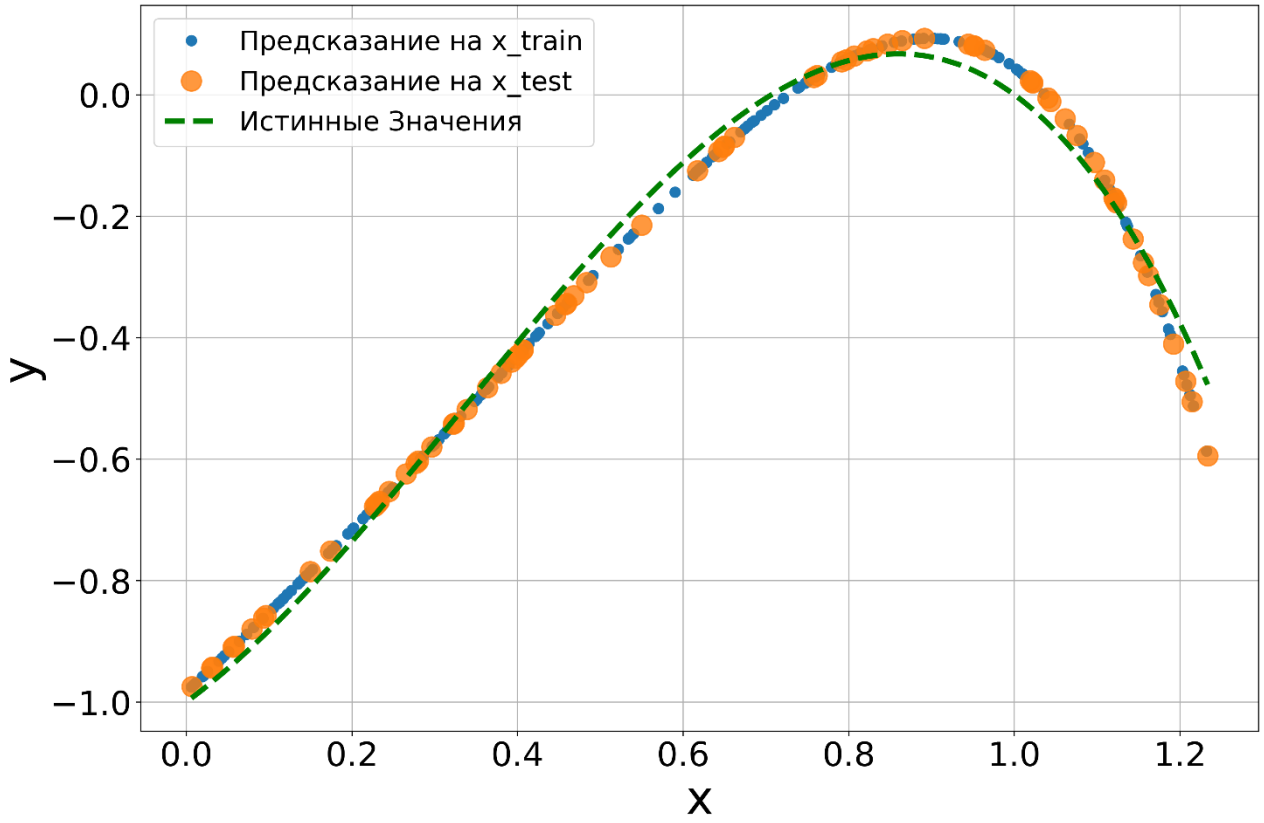


Рис. 6-9 Полученная линейная модель для полиномиальной зависимости с использованием полиномиальных признаков

Использование полиномов от исходных признаков – достаточно простой, но работающий метод инженерии признаков, который можно использовать на начальном этапе работы с данными. Можно считать, что степень полиномов – тоже гиперпараметр модели линейной регрессии. Однако, как и любой другой инструмент его нужно использовать с осторожностью. В частности – какой степени полинома будет достаточно.

Регуляризация линейной регрессии

Регуляризация Тихонова

Как было написано ранее бывает так, что обычный градиентный спуск приводит к переобучению модели. Переобучение — это ситуация, когда точность на обучающих данных значительно выше, чем на тестовых. В таких случаях также можно сказать, что данные плохо обусловлены - то есть любые небольшие изменения по отношению к тренировочной выборке приведут к

большим изменениям в ответе модели. В целом это будет означать, что модель дает очень большой разброс результатов.

Такой разброс может быть снижен при помощи различных техник регуляризации. Смысл использования таких техник сводится к тому, что при обучении модели к выражению обновления весовых параметров добавляется дополнительное условие. Например, можно добавить условие ограничение суммы квадратов весовых параметров. Такое предположение называется регрессией Тихонова или гребневой регрессией (а также L2 регуляризацией).

Технически такая регуляризация соответствует предположению, что распределение результатов работы модели имеет вид нормального распределения. Такое предположение часто допустимо и оправдано. Функция потерь для регрессии Тихонова может быть записана в следующей форме:

$$\{L(\hat{y}_i, y_i) \to min \;\; \| W\|_2^2 < const \;\to L(\hat{y}_i, y_i) + \frac{\lambda}{2}\sum_{j=1}^{p} W_j^2 \to min,$$

где $\|W\|_2^2 = \sum_{j=1}^{p} W_j^2$ - норма Фробениуса для вектора или матрицы; λ - регуляризационный множитель; p - размер вектора весовых параметров; N - объем выборки весовых параметров. Также отметим, что смещение не регуляризуется. Как правило регуляризационный множитель задается в диапазоне от 0 до 0.1, часто проверяются значения в логарифмическом масштабе (0.1, 0.01, 0.001 и т.д).

Оптимизацию с использованием регуляризации можно интуитивно представить в виде следующего графика на Рис. 6-10.

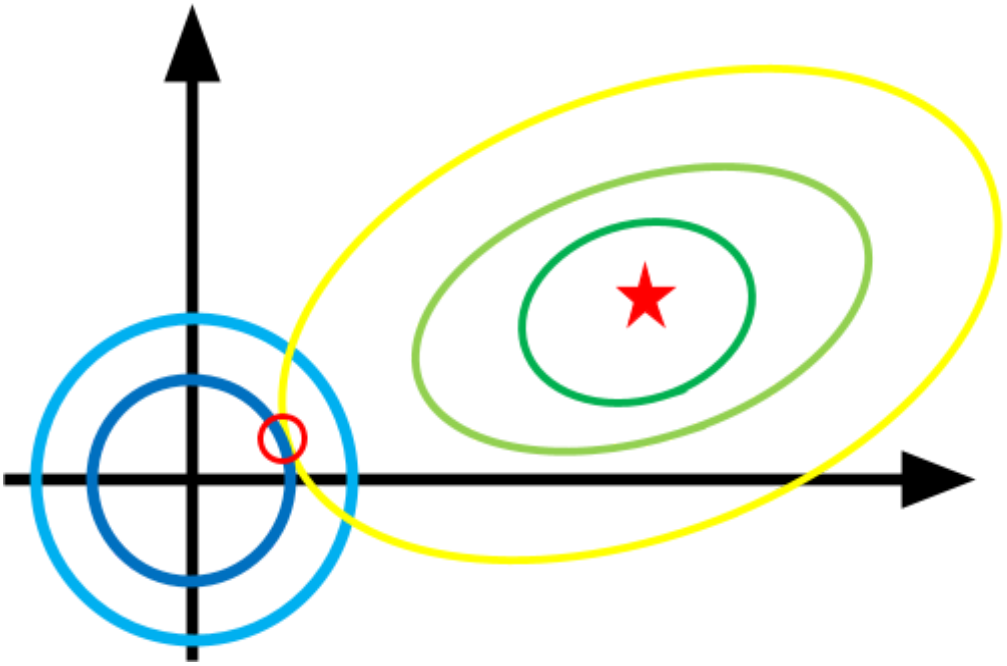


Рис. 6-10 Схематичное изображение L2-регуляризации

Звезда на графике обозначает минимум функции потерь для линейной регрессии. Желтыми и зелеными эллипсами схематично отображаются комбинации весов, при которых функция потерь принимает одинаковое значение. Окружности в центре координат символизируют ограничения, задаваемое регуляризации. Таким образом при регуляризации находится своеобразный компромисс: мы хотим, чтобы веса были на окружности наименьшего радиуса и при этом функция потерь принимала как можно меньшее значение. При этом «удельный вес» вклада регуляризации и функции потерь задается регуляризационным множителем.

Закон изменения весовых параметров для данной модели можно записать как:

$$W^t = W^{t-1} - \eta\left(\frac{1}{N}\sum_{i=0}^{N-1}\nabla_W L\left(\hat{y}_i, y_i\right) + \lambda\sum_{j=1}^{p} W_j\right).$$

Запишем новую версию регрессии RidgeRegression на основе уже созданного класса LinearRegression. Для этого запишем новый класс, наследующий от уже созданного, и перепишем в нем методы .loss и .update.

Регуляризация L1

В ряде случаев, когда разброс данных оказывается очень большим, регуляризации L2 может оказаться бесполезной или даже вредной. Дело в том, что в функции потерь мы учитываем веса в квадрате и большие колебания весовых параметров в квадрате приведут к большим колебаниям в значениях функции потерь. Часто эта ситуация является недопустимой.

В таких случаях следует выбирать более устойчивые (робастные методы). Робастные методы могут быть менее точными, однако более стабильными. Одним из наиболее распространённых робастных методов является L1 регуляризация (или регуляризация Лассо). В этом случае выражение для функции потерь может быть записано следующим образом:

$$\{L(\hat{y}_i, y_i) \rightarrow min \;\; \| W\|_1^1 const \;\rightarrow L(\hat{y}_i, y_i) \; + \; \lambda \sum_{j=1}^{p} W_j \rightarrow min,$$

где $\|W\|_1^1 = \sum_{j=1}^{p} W_j$ - норма L1 для вектора или матрицы; λ - регуляризационный множитель; p - размер вектора весовых параметров; N - объем выборки весовых параметров.

Изменение L2 нормы на L1 приводит к достаточно интересному эффекту Рис. 6-11.

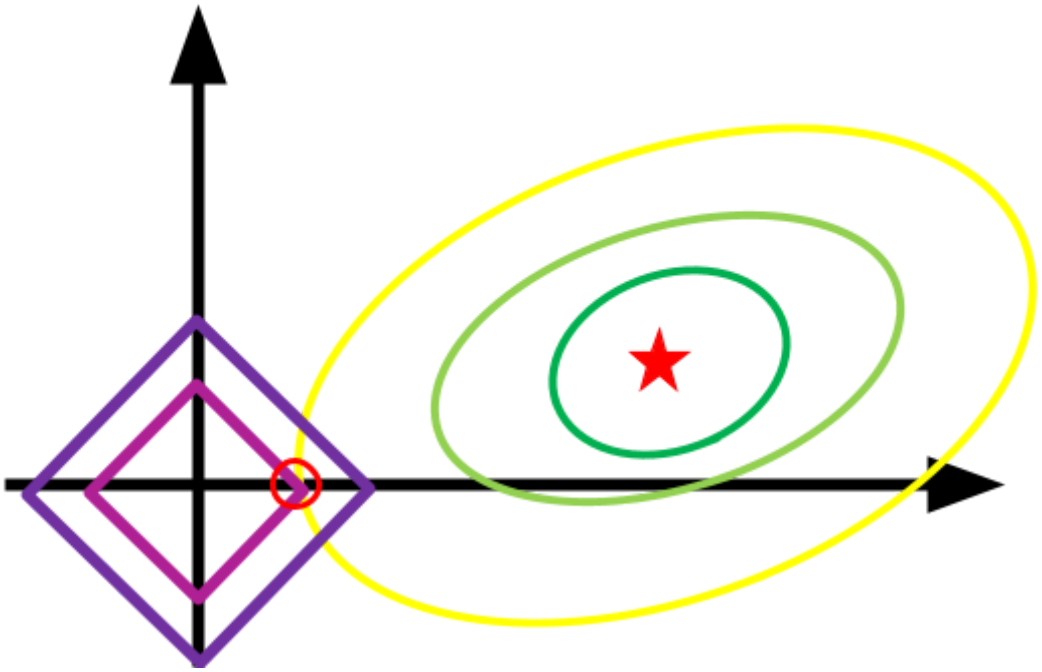


Рис. 6-11 Схематичное изображение L1-регуляризации

Теперь вместо ограничивающей окружности – ограничивающий ромб. Что интересно такой «угловатый» ограничитель приводит к тому, что для некоторых весов компромисс регуляризации попадет на ось координат. Это в свою очередь приводит к тому, что часть весов обнулится. А это приведет к тому, что от части признаков, модель избавится самостоятельно, что бывает полезно в случае анализа данных с большим числом признаков.

Закон изменения весовых параметров для данной модели можно записать как:

$$W^t = W^{t-1} - \eta\frac{1}{N}\sum_{i=0}^{N-1}\nabla_W L(\hat{y}_i, y_i) + \lambda.$$

Запишем новую версию регрессии LassoRegression на основе уже созданного класса LinearRegression. Для этого запишем новый класс, наследующий от уже созданного, и перепишем в нем методы .loss и .update.

Эластичная регуляризация

Отметим, что во многих случаях неизвестно какая модель регуляризации окажется лучше, поэтому целесообразно использовать их обе вместе. Такая модель регуляризации называется эластической регуляризацией. Запишем новую версию регрессии ElasticRegression на основе уже созданного класса LinearRegression. Для этого запишем новый класс, наследующий от уже созданного, и перепишем в нем методы .loss и .update.

Как правило имеет смысл протестировать различные варианты регуляризации, так и различные значения регуляризационных множителей. Все это также можно отнести к гиперпараметрам модели линейной регрессии.

Ссылки на справочные материалы

1. Блог с описанием ключевых терминов и понятий машинного обучения «простыми словами». *Блок про регрессию* https://vas3k.ru/blog/machine_learning/
2. Описание линейных моделей https://ml-handbook.ru/chapters/linear_models/intro#%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F-%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D1%8F-%D0%B8-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-%D0%BD%D0%B0%D0%B8%D0%BC%D0%B5%D0%BD%D1%8C%D1%88%D0%B8%D1%85-%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BE%D0%B2-%D0%BC%D0%BD%D0%BA
3. Описание метрик регрессии https://ml-handbook.ru/chapters/model_evaluation/intro#%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D1%8F

Примерные Вопросы для контроля

1. Каковы ключевые части метода наименьших квадратов для поиска коэффициентов линейной регрессии? (вопрос-дискуссия)
2. В чем заключаются основные различия между методом наименьших квадратов и градиентным спуском для нахождения коэффициентов регрессии? (вопрос-дискуссия)
3. Может ли коэффициент детерминации быть отрицательным числом? Если «да» - в каких случаях, если «нет» - почему? (Может. Если модель предсказывает хуже, чем среднее значение по выборке)
4. Почему L1-регуляризация может привести к отбору признаков (в отличие от L2-регуляризации)? (вопрос-дискуссия)
5. Почему регуляризация помогает уменьшить переобучение? (вопрос-дискуссия)

Классификация

Список тем, которые должны быть осуждены на лекции:

1. Классификация
2. Логистическая регрессия
3. Метрики Классификации

Ключевые моменты по темам:

1. Задача классификации. Обсуждение примеров задач из реальной жизни, которые можно свести к классификации.
   Типы классов (бинарная и мультиклассовая классификация, пересекающиеся и непересекающиеся классы, нечеткие классы)
   Области применения задач классификации (дискуссия)
2. Логистическая регрессия
   Переход от модели линейной регрессии для решения задачи классификации
   Функция потерь логистической регрессии
   Градиентный спуск и регуляризация модели логистической регрессии
   Стратегии один против всех и один против одного для мультиклассовой классификации
   SWOT-анализ логистической регрессии
3. Обсуждение метрик классификации

Подведение Итогов Лекции

Текстовые материалы

Один из самых простых методов классификации — это логистическая регрессия. По существу, модель логистической регрессии представляет собой аналог линейной регрессии.

Интересный исторический факт. Был момент, когда логистическая регрессия использовалась для классификации только одного: если вы выпьете

пузырек с ядом, вас, скорее всего, назовут «живым» или «умершим»? Времена изменились. Сегодня вызов экстренных служб дает лучший ответ на этот вопрос, а логистическая регрессия лежит в основе глубокого обучения.

Борьба с ядами. Логистическая функция восходит к 1830-м годам, когда бельгийский статистик П.Ф. Ферхюльст изобрел его для описания динамики населения: со временем первоначальный взрыв экспоненциального роста сглаживается по мере того, как он потребляет доступные ресурсы, что приводит к характерной логистической кривой. Прошло более века, прежде чем американский статистик Э. Б. Уилсон и его ученица Джейн Вустер разработали логистическую регрессию, чтобы выяснить, какое количество данного опасного вещества может привести к летальному исходу.

Логистическая функция описывает широкий спектр явлений с достаточной точностью, поэтому логистическая регрессия обеспечивает полезные базовые прогнозы во многих ситуациях. В медицине она оценивает смертность и риск заболевания. В политической науке она предсказывает победителей и проигравших на выборах. В экономике она прогнозирует перспективы бизнеса. Что еще более важно, она управляет частью нейронов, в которых нелинейность является сигмовидной, в самых разных нейронных сетях.

Генерируемые данные

Как и в случае с линейной регрессии давайте сначала сгенерируем данные. В этом учебном пособии мы рассмотрим только простой вариант классификации – Бинарная классификация. В этот раз мы воспользуемся готовыми функциями, которые реализованы в библиотеки scikit-learn, а именно линейно-разделимые данные (make_classification), данные, распределенные в виде знака «Инь-Ян» (make_moons) и концентрические круги (make_circles).

Рис. 6-12 Типы генерируемых данных

Запишем отдельную единую функцию, которая будет генерировать данные. Входными параметрами этой функции будет N – количество точек, method – тип данных, который мы хотим сгенерировать, noises – уровень шума в данных. Выходными параметрами будут признаки – матрица X с размерами $N\times 2$, и вектор y – меток классов для каждой точки.

```
from sklearn.datasets import make_moons, make_circles, make_classification

def make_bin_clf(N, method = 'line', noises = 0.15, random_state = 42):

    if random_state: rng = np.random.RandomState(seed = random_state)

    if method == 'line' or method is None:
        X, y = make_classification(n_samples=N, n_features=2, n_redundant=0, n_informative=2,
n_clusters_per_class=1, class_sep=2)
        X += np.random.randn(*X.shape) *noises

    elif method == 'moons':
        X, y = make_moons(n_samples=N, noise=noises)

    elif method == 'circles':
        X, y = make_circles(n_samples=N, noise=noises, factor=0.5)

    return X,y
```

Для простоты начнем работу с набором данных, который линейно разделяем.

Модель логистической регрессии

В случае линейной регрессии мы хотели провести линию через наши данные, таким образом, чтобы в среднем отклонение точек было минимальным. С точки зрения бинарной классификации нам необходимо предсказывать всего два числа, допустим это 0 и 1. Теоретически мы можем использовать модель линейной регрессии, но есть фундаментальная загвоздка: в общем случае линейная регрессия выдает ответ в диапазоне $(-\infty, \infty)$. Да, наверняка с помощью градиентного спуска мы, наверное, сможем подобрать такие веса, которые будут давать необходимый ответ. Но давайте поможем нашей модели – обернем ее в некоторую функцию, которая преобразует данные в нужный нам диапазон. Примером такой функции является функция сигмоида, или логистическая функция, которую мы упоминали ранее в главе 1. Эта функция описывается следующим уравнением

$$\sigma(z_i) = \frac{1}{1+exp(-z_i)}$$

На рисунке Рис. 6-13 представлен график функции сигмоиды на интервале от -10 до 10.

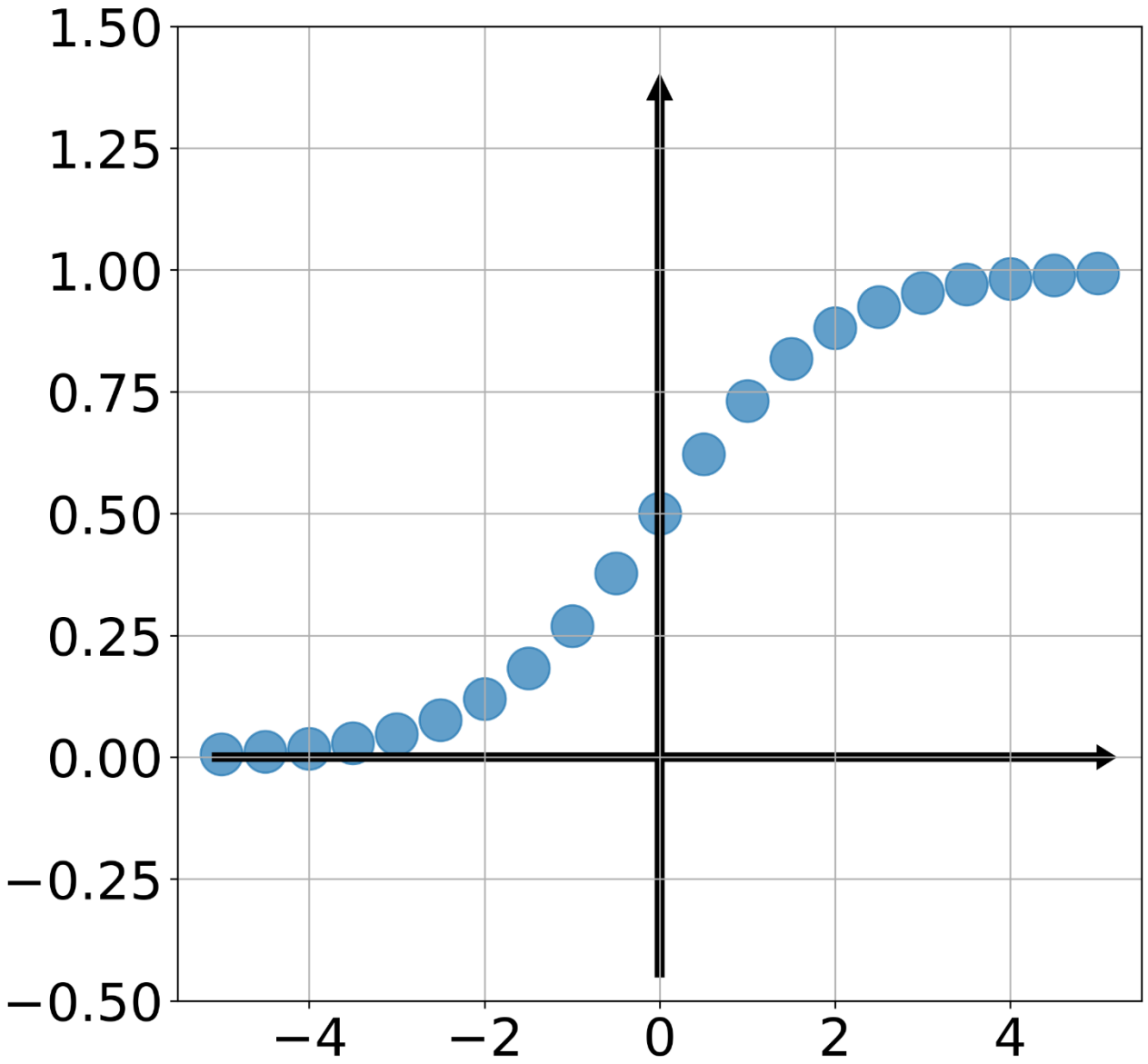


Рис. 6-13 График функции сигмоиды

В Python реализовать сигмоиду тоже достаточно просто.

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

Итого мы получили следующую модель, которая представляет собой логистическую функцию, которая применяется к модели линейной регрессии. Отсюда название модели – логистическая регрессия, не смотря на то что решается задача классификации. Функционально модель можно описать следующим образом:

$$\hat{y}_i = \sigma\left(\sum_{j=0}^{p} w_j X_{ij}\right) \equiv \sigma\left(\sum_{j=1}^{p} w_j X_{ij} + b\right) = \sigma(z_i),$$

где σ - функция сигмоиды; $\hat{y}_i$ - результат принятия решений, $z_i = \sum_{j=0}^{p} w_j X_{ij}$.

Из некоторых статистических выводов известно, что для такой модели необходимо выбрать функцию потерь следующего вида:

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=0}^{N-1}[y_i log\hat{y}_i + \left(1 - y_i\right)log\left(1 - \hat{y}_i\right)]$$

Эта функция потерь называется бинарная кросс-энтропия.

Прежде чем записать выражение для градиента функции потерь запишем выражение для производной функции активации

$$\sigma'(z_i) = \frac{\partial\sigma(z_i)}{\partial z_i} = (1 - \sigma(z_i))\sigma(z_i)$$

Градиент функции потерь для одного элемента выборки может быть выражен следующим образом:

$$\nabla_W L_i = -\left(\frac{y_i}{\sigma(X_i W^T)} - \frac{1-y_i}{1-\sigma(X_i W^T)}\right)\sigma'(X_i W^T)\odot X_i = -\left(y_i - \sigma(X_i W^T)\right)\odot X_i = -\left(y_i - \hat{y}_i\right)$$

Тогда правило обновления весовых параметров может быть записано как

$$W^t = W^{t-1} - \eta\frac{1}{n}\sum_{i=0}^{n-1}\left(\hat{y}_i - y_i\right)\odot X_i.$$

Отметим, что данное выражение эквивалентно записанному для линейной регрессии с точностью до коэффициента 2, поэтому мы учтем данный параметр путем замены $\eta \to \eta/2$.

Как правило, после расчета функции сигмоиды мы должны округлить значения до 0 или до 1. То есть до значения метки одного из классов. Такое округление можно сделать по заданному порогу. Например, мы можем сказать, что, если значение сигмоида больше 0.5, то пусть будет класс 1, а если меньше, то наоборот. Однако, на практике иногда ставят высокий порог, 0.7 - 0.8.

Запишем функцию определения класса.

```
def to_class(logit, threshold = 0.7):
    return (logit>=threshold)*1
```

Отметим, что в описанном смысле можно говорить о том, что результат сигмоида — это вероятность того, что аргумент функции $\sigma(z_i)$ принадлежит одному из классов. Также отметим, что такой аргумент принято называть логит. Отметим, что для расчета функции потерь не следует пользоваться округлением до классов

Теперь запишем функцию потерь. Отметим, что значениях логарифма мы ввели небольшую константу eps с целью исключить ошибку вида «логарифм нуля».

```
def bce_loss(yhat, y):
    eps = 1e-6
    return -(y*np.log(yhat + eps)+(1-y)*np.log(1-yhat+eps)).mean()
```

Запишем все в один класс LogisticRegression. Однако не будем писать класс с нуля, а создадим его на основе имеющего класса ElasticRegression. Это позволит не писать методы с нуля, а использовать готовые наработки. Это адекватно, потому что градиентный спуск применяется одинаково как в случае линейной регрессии и логистической регрессии, с точки зрения самого алгоритма. Меняется модель и функция потерь.

Дополнительно в классе LogisticRegression реализован метод .plot_desicion_function, который позволяет визуализировать в двумерном пространстве вероятность принадлежности отдельной точки к конкретному классу – функция принятия решений. На Рис. 6-14 представлена визуализация функция принятия решений для линейно разделимых данных.

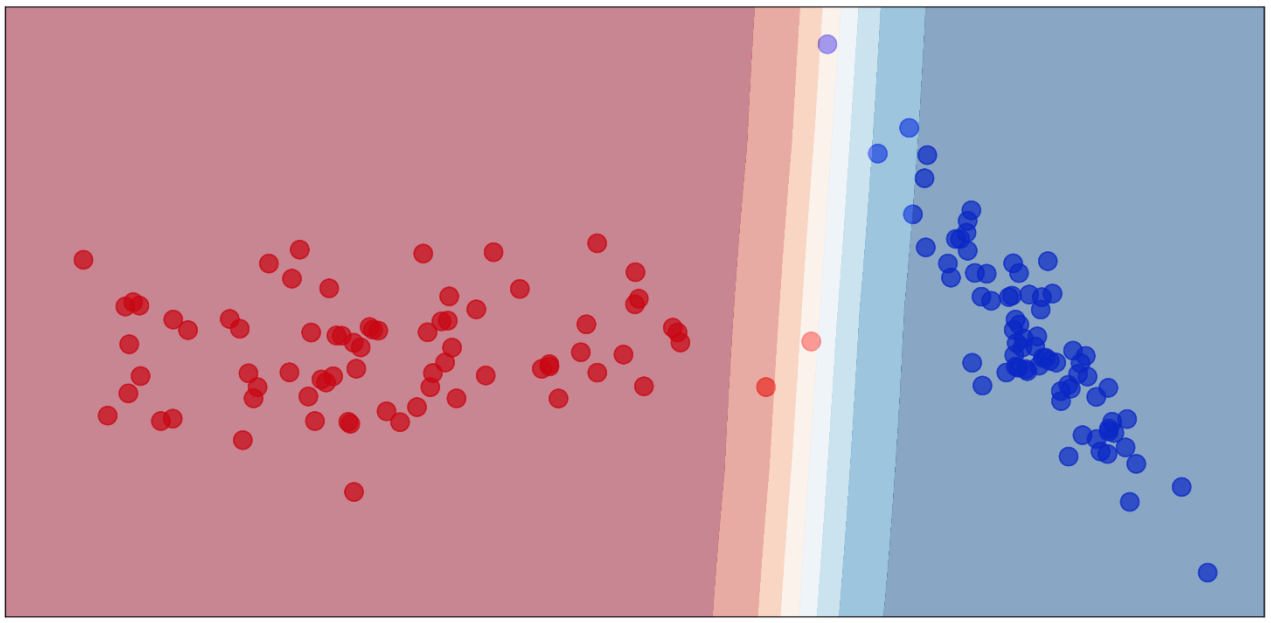


Рис. 6-14 Визуализация функции принятия решений для Линейно разделимых данных

Как и ожидалось Логистическая регрессия разделяет классы с помощью линий, которые проводятся таким образом, чтобы минимизировать функцию потерь. Однако это не всегда возможно сделать. Так, для того чтобы классифицировать набор данных концентрические круги необходимо

предварительно сгенерировать полиномиальные признаки, чтобы усложнить модель. Пример успешной классификации концентрических кругов с помощью логистической регрессии и полиномиальных признаков до 2 степени представлен на Рис. 6-15.

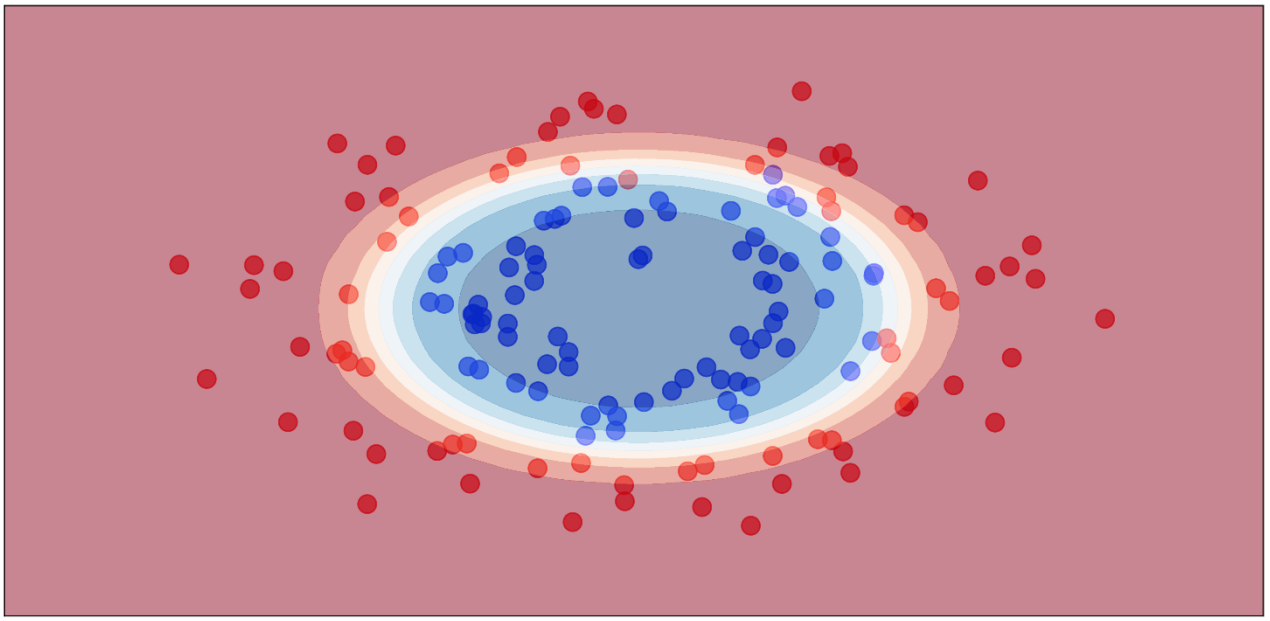


Рис. 6-15 Визуализация функции принятия решений для концентрических кругов

Дополнительно в классе LogisticRegression, реализован метод .classification_report, в котором представлен расчет различных метрик классификации. Давайте остановимся на них отдельно

Метрики классификации

Рассмотрим пример ошибок классификации (Рис. 6-16). У нас есть два класса, красный и зеленый, при этом красный класс - основной. Идеальная модель – зеленая, она отделяет все красные точки от зеленых. Но допустим в ходе обучения у нас получилась красная модель. И есть два типа ошибок, которые совершила красная модель. К ошибкам 1 рода относят неправильно отнесенные к основному классу объекты: те зеленые, которые наша модель ошибочно считает красными. К ошибкам второго рода относят пропуски, те красные точки, которые не вошли в предсказании модели.

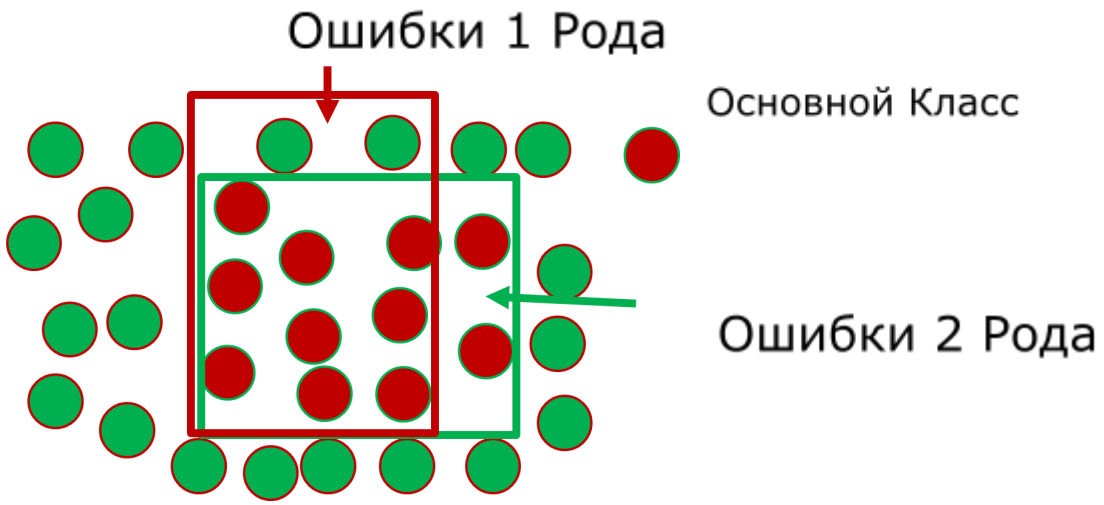


Рис. 6-16 Ошибки классификации

Соотношения между правильными предсказаниями, ошибками первого и второго рода и лежат в основе оценки моделей классификации. Для удобства заполняется матрица ошибок (confusion matrix), пример которой представлен на Рис. 6-17. В советующие ячейки матрицы вставляются количество правильно предсказанных объектов основного класса (True Positive – TP), количество правильно предсказанных объектов не основного класса (True Negative – TN), количество ошибок 1 рода (False Positive – FP), количество ошибок 2 рода (False Negative).

Рис. 6-17 Мат

	$y = 1$	$y = 0$
$\hat{y} = 1$	True Positive (TP)	False Positive (FP)
$\hat{y} = 0$	False Negative (FN)	True Negative (TN)

За ки классификации:

а

$$precision = \frac{TP}{TP+FP}$$

- **Recall** Полнота

$$recall = \frac{TP}{TP+FN}$$

- **Specificity** Специфичность

$$specificity = \frac{TN}{TN+FP}$$

- **F1**-мера

$$F_1 = \frac{2*precision*recall}{precision+recall}$$

Наиболее очевидная метрика – доля правильных ответов хорошо работает в случае сбалансированных классов, т. е. когда число объектов разных классов совпадает. Однако в случае несбалансированных классов, эта метрика может вводить в заблуждение: допустим у нас два класса, в первом классе 900 объектов, во втором 100. Если модель будет предсказывать все объекты как первый класс, то доля правильных ответов будет 0.9 или 90%, что вроде неплохо. При этом второй класс модель совсем не предсказывает.

Для уточнения предсказательной способности модели вводят дополнительные метрики, которые оценивают долю ошибок первого и второго рода отдельно (Точность, Полнота, Специфичность). Иногда для удобства используют F-1 меру, среднее гармоническое между Точностью и Полнотой.

В классе LogisticRegression для оценки модели классификации реализован метод .score, который вычисляет метрику **Accuracy**.

Ссылки на справочные материалы

1. Блог с описанием ключевых терминов и понятий машинного обучения «простыми словами». *Блок про классификацию* https://vas3k.ru/blog/machine_learning/
2. Описание линейных моделей https://ml-handbook.ru/chapters/linear_models/intro#%D0%BB%D0%BE%D0%B3%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%8

1%D0%BA%D0%B0%D1%8F-%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D1%8F

3. Описание метрик классификации https://ml-handbook.ru/chapters/model_evaluation/intro#%D0%B1%D0%B8%D0%BD%D0%B0%D1%80%D0%BD%D0%B0%D1%8F-%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F-%D0%BC%D0%B5%D1%82%D0%BA%D0%B8-%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2

Примерные Вопросы для контроля

1. В чем основное различие между задачами классификации и задачами регрессии? (вопрос-дискуссия)
2. В чем разница между применением градиентного спуска в линейной регрессии и логистической регрессии? (вопрос-дискуссия)
3. Приведите пару примеров задач бинарной классификации и мультиклассовой классификации. (вопрос-дискуссия)

7. Методы Разложение Матриц

Список тем, которые должны быть осуждены на лекции:

1. Уменьшение Размерности
2. Ковариационная Матрица
3. Метод Главных Компонент
4. Сингулярное разложение

Ключевые моменты по темам:

1. Задача уменьшения размерности. Обсуждение примеров в которых пригодится уменьшение размерности.
   Подход к уменьшению размерности основанный на распределении данных. Необходимость стандартизации данных
2. Определение ковариации, матрицы ковариации.
   Расчет матрицы ковариации для стандартизованных данных
3. Метод главных компонент. Поиск собственных значений и собственных векторов матрицы ковариации
   Объясненная дисперсия, кумулятивная объясненная дисперсия – анализ собственных значений матрицы ковариации
   Главные компоненты – линейная комбинация исходных признаков, коэффициенты в этой комбинации – собственные вектора
   Подходы к выбору уменьшенного пространства признаков (для визуализации, исходя из значений кумулятивной объясненной дисперсии, порога объясненной дисперсии)
   Ключевые шаги метода главных компонент
4. Представление метода главных компонент как разложения квадратных матриц.
   Сингулярное разложение матриц, как обобщенный случай разложения неквадратных матриц.
   Матрицы U, S, V
   Варианты сингулярного разложения матриц: базовый вариант, экономный вариант, компактный вариант, усеченный вариант.

Сжатие данных с использованием сингулярного разложения матриц

Получение псеводообратной матрицы с использованием сингулярного разложения матриц

SWOT-анализ метода главных компонент и сингулярного разложения матриц

Подведение Итогов Лекции

Текстовые материалы

В этой главе мы рассмотрим базовый подход к уменьшению размерности, который относится к методу разложения матриц - Метод главных компонент (Principal Components Analysis). Это метод снижения размерности данных путем преобразования их в такую форму, чтобы оставить только максимально полезную информацию. Под термином полезная информация в данном методе понимается набор независимых друг от друга признаков с максимальной дисперсией (с максимальным разбросом значений). При этом мы изначально полагаем, что интенсивность (в некотором смысле) полезной информации преобладает над интенсивностью каких-то случайных искажений или других не идеальностей нашего набора данных.

Для выделения полезной информации в методе главных компонент проводится преобразование данных от набора исходных столбцов (исходных признаков), которые могут содержать шумы и быть линейно зависимыми к набору новых столбцов, которые обладают важным свойством линейной независимости (не коррелируют). Такие новые столбцы можно изобразить как некоторую систему координат, в которых можно отложить точки - наши данные. При этом часто оказывает так, что некоторые из координат не нужны - в них почти наверняка нет информации.

Генерируемые данные

Как обычно, рассмотрим пример на данных, структуру которых мы можем контролировать. Это можно представить, как если мы возьмём двухмерную

фигуры эллипс с разными радиусами (Рис. 7-1 а). При этом окажется, что можно выбрать такую систему координат, что вторая ось «не нужна». При этом даже если мы возьмём двумерную цельную фигуру, но по одной из осей у нас будет «совсем чуть-чуть», то можно рассмотреть этот случай как одномерный (Рис. 7-1 б).

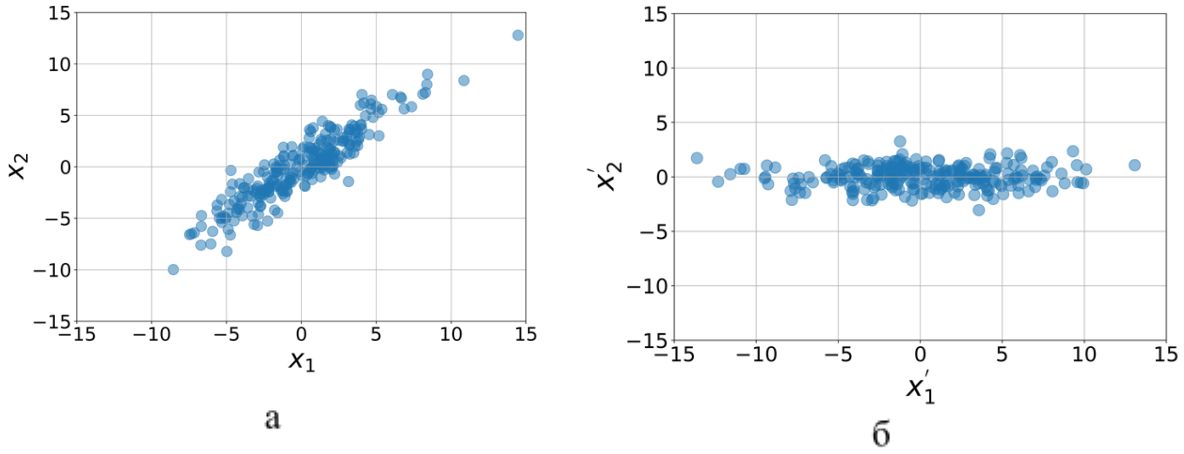


Рис. 7-1 Уменьшение размерности двумерных данных

Сгенерировать данные, распределенные подобным образом можно используя следующую функцию create_elipsoid_data, где C1 и C2 – координаты центра, S1 и S2 радиусы эллипса, theta угол наклона, N количество точек, random_state

```
def create_elipsoid_data(C1 = 0,C2 = 0 ,S1 = 5,S2 =1, theta =45, N = 250, random_state = 42):
    if random_state: np.random.seed(random_state)

    theta = np.pi*theta/180
    Centers = np.array([C1,C2])
    Sigmas = np.array([S1,S2])

    R = np.array([[np.cos(theta), - np.sin(theta)],
                  [np.sin(theta), np.cos(theta)]])
```

```
return (R @ np.diag(Sigmas) @ np.random.randn(2, N)+np.diag(Centers)@ np.ones((2, N))).T
```

В нашем примере одна ось, в которой будет основная часть фигуры, будет главной компонентой. Совокупность наших главных компонент образует т. н. собственное подпространство. Вторая ось останется т. н. шумовым подпространством. Как видно из примера на Рис. 7-1 б размерность фигуры в каждой из обозначенных осей будет соответствовать важности этой оси.

Другими словами, можно сказать, что разброс значений в каждой оси будет соответствовать ее важности. Такой разброс значений по каждой оси будет называться - собственные значения. Сортируя собственные значения по убыванию, мы можем определить те из них, которые следует оставить и те, которые следует убрать.

В случае данных большей размерности происходят схожие преобразования: находятся оси многомерных эллипсов, в которых данные изменяются наибольшим образом. При этом уменьшение размерности можно реализовать за счет выбора не всех главных компонент, а только те, которые имеют наибольшее собственное значение.

Метод главных Компонент

Формализуем алгоритм вычисления главных компонент. Для этого нам нужно ввести дополнительные понятия и определения. **Матрица ковариации** — это матрица, составленная из попарных ковариаций столбцов этой матрицы. **Ковариация** — это мера совместной изменчивости двух случайных величин.

Пусть у нас есть матрица данных $X \in R^{n \times p}$, где p – количество признаков, n – количество точек. В общем случае, для определения матрицы ковариации необходимо воспользоваться следующей формулой

$$\Sigma = cov(X, X) = E[(X - E[X])^T (X - E[X])] ,$$

где E - оператор математического ожидания.

Однако эту формулу можно упростить, предварительно выполнив централизацию матрицы X, т. е. вычесть среднее для каждого признака. Тогда вычисление матрицы ковариации просто сведется к матричному умножению транспонированной центрированной матрицы на саму себя. При этом при анализе реальных разнородных данных признаки не просто центрируют, а стандартизируют, добиваясь одинаковой дисперсии признаков. Это оправдано, когда сопоставляются признаки, измеряемые в разных единицах измерений (например, возраст в годах и зарплата в рублях).

В результате получится матрица $\Sigma \in R^{p \times p}$, т. е. квадратная матрица. Отметим, что собственные вектора и собственные значения для произвольной квадратной матрицы A удовлетворяют следующему уравнению:

$$A\vec{v}_i = \lambda_i \vec{v}_i,$$

где $\vec{v}_i$ — это собственный вектор; λ_i - соответствующее собственное значение. По сути, выражение выше представляет собой решение системы линейных уравнений с параметром λ_i. Значения данного параметра можно найти из следующего выражения:

$$det(A - \lambda_I) = 0,$$

где det - операция поиска определителя матрицы, а λ_I - диагональная матрица с собственными значениями по главной диагонали и нулями в остальных позициях. При раскрытии операции детерминанта по определению данное уравнение может быть сведено к поиску корней полинома.

Цель Метода Главных компонент найти и отбросить шумовое подпространство. Классический метод главных компонент состоит из следующих операций:

1. вычисление ковариационной матрицы для набора данных - то есть матрицу дисперсий.
2. вычисление (поиск) собственных векторов и их собственных значений по ковариационной матрице.
3. сортировка собственных значений по убыванию.
4. выделения собственного подпространства.
5. преобразование данных - построение проекции исходного массива на полученные собственные вектора.

Все эти операции реализованы в классе PCA, который представлен в Приложении 3. Размерность собственного пространства задается входным параметром n_components. При этом операции с 1 по 4 реализованы методом .fit. Последняя операция реализована методом .transform. Также реализована операция .inverse_transform, необходимая для восстановления исходного набора данных. Собственные значения для матрицы данных хранятся в атрибуте values, а собственные вектора – в атрибуте components. Дополнительно в классе PCA реализован метод .plot_eigvalues для визуализации собственных значений.

Отметим, что, по сути, пространство главных компонент является линейной комбинацией всех исходных признаков, при этом веса в этой линейной комбинации определяются элементами собственных векторов. Соответствующие собственные значения показывают значимость главной компоненты. Анализируя распределение собственных значений можно делать выводы о том, сколько главных компонент вносят основной вклад в дисперсию исходных данных.

Важно отметить, что на практике такое восстановление может быть не точным так как сокращая разность данных вы можете удалить оттуда и часть полезной информации. Для этого в классе PCA реализован метод .score, который использует метрику коэффициент детерминации, который мы обсуждали в главе 3.

Наконец обсудим применение метода главных компонент для категориальных признаков. Существует разные взгляды на то, возможно ли это и насколько это корректно. Некоторые работы показывают, что достаточно применить one-hot кодирование к категориальным признакам, а затем применить метод главных компонент «как есть». С другой стороны, авторы метода Факторного Анализа Смешанных Данных (Factorial Analysis of Mixed Data FAMD) считают, что их подход обобщает метод главных компонент на случай категориальных признаков. Для необходимо сначала выполнить преобразование one-hot кодирование. Затем нормировать каждый столбец закодированных категориальных признаков, чтобы соотнести между собой категориальные и числовые признаки. Для этого мы делим каждый столбец на корень из вероятности признака после one-hot кодирования (количество единиц в столбце, деленное на количество строк). Деление на квадратный корень из вероятностей можно интерпретировать так: мы придаем большую вес редким признакам, потому что информация «когда этот признак встречается» более значима, чем когда признак очень редкий. А затем метод главных компонент применяется «как обычно».

Набор данных MNIST

Возможно, поиск осей в двумерном эллипсе не выглядит как что-то впечатляющее. Посмотрим возможности метода главных компонент на наборе данных MNIST. Набор MNIST – набор рукописных чисел от 0 до 9. Общее число изображений в наборе данных – 70000. Пример изображений, которые содержатся в этом наборе данных представлены на Рис. 7-2.

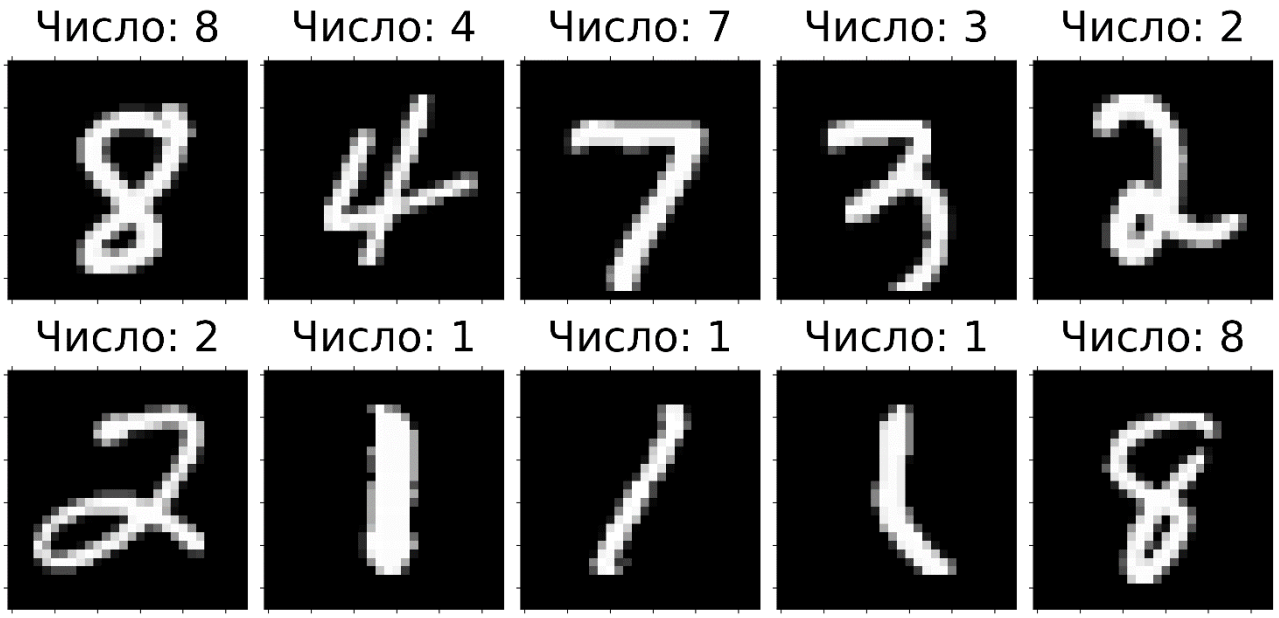


Рис. 7-2 Примеры изображений в наборе данных MNIST

Изображения в наборе MNIST – одноканальные, размером 28 на 28 пикселей. Воспользуемся простым подходом и будем рассматривать эти изображения как вектор признаков из 784 параметров. Для загрузки данных мы воспользуемся функцией fetch_openml для загрузки данных с сайта OpenML используя имя набора данных 'mnist_784'. Помимо набора данных MNIST сайт OpenML содержит большое число других наборов данных, которые можно использовать для практики алгоритмов машинного обучения.

В коде ниже мы также указали, что хотим чтобы функция вернула нам только массив данных и вектор меток (return_X_y=True), а не полные данные в формате словаря. При этом мы указали, чтобы данные сразу были в формате массивов numpy array, а не датафреймов Pandas (as_frame = False).

```
from sklearn.datasets import fetch_openml

X, y  = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame =
False)
```

Данные в наборе MNIST хранятся в формате 8 бит (т. е. интенсивность пикселя закодирована числом от 0 до 255). Нормализуем данные, для этого набора данных достаточно поделить все значения на 255. При этом данные будут изменяться в диапазоне от 0 до 1, где 0 – это черные пиксели, а 1 – белые. Промежуточные значения – 254 оттенка от черного к белому.

Воспользуемся подготовленным нами классом PCA и применим метод главных компонент к набору данных MNIST, указав n_components равным 100. На Рис. 7-3представлена визуализация собственных значений для проведенного преобразования. Видно, что после 100 собственного значения график асимптотически стремится к 0.

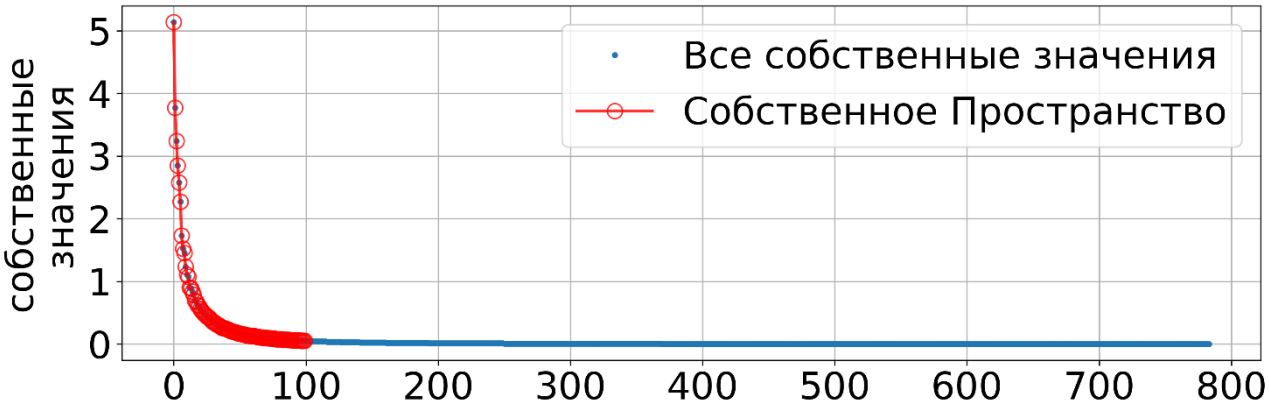


Рис. 7-3 Визуализация собственных значений для набора данных MNIST

На Рис. 7-4 представлена визуализация скатерограмм для первых 4 главных компонент. Каждая цифра представлена точкой соответствующего цвета. Удивительно, но несмотря на ограниченную размерность ряд цифр сгруппированы «по группам». Так цифры 0 и 1 стоят обособленно от остальных в координатах 1 главной компоненты, 2 главная компонента позволяет отделить цифру 2 от остальных, а 4 главная компонента позволяет «отсечь» часть цифр 6. При этом «перемешанными» оказались цифры 3 и 5, а также цифры 4, 7 и 9. Это вполне объяснимо, из-за схожести в написании этих цифр. Тем не менее это достаточно хороший результат для такой простой модели.

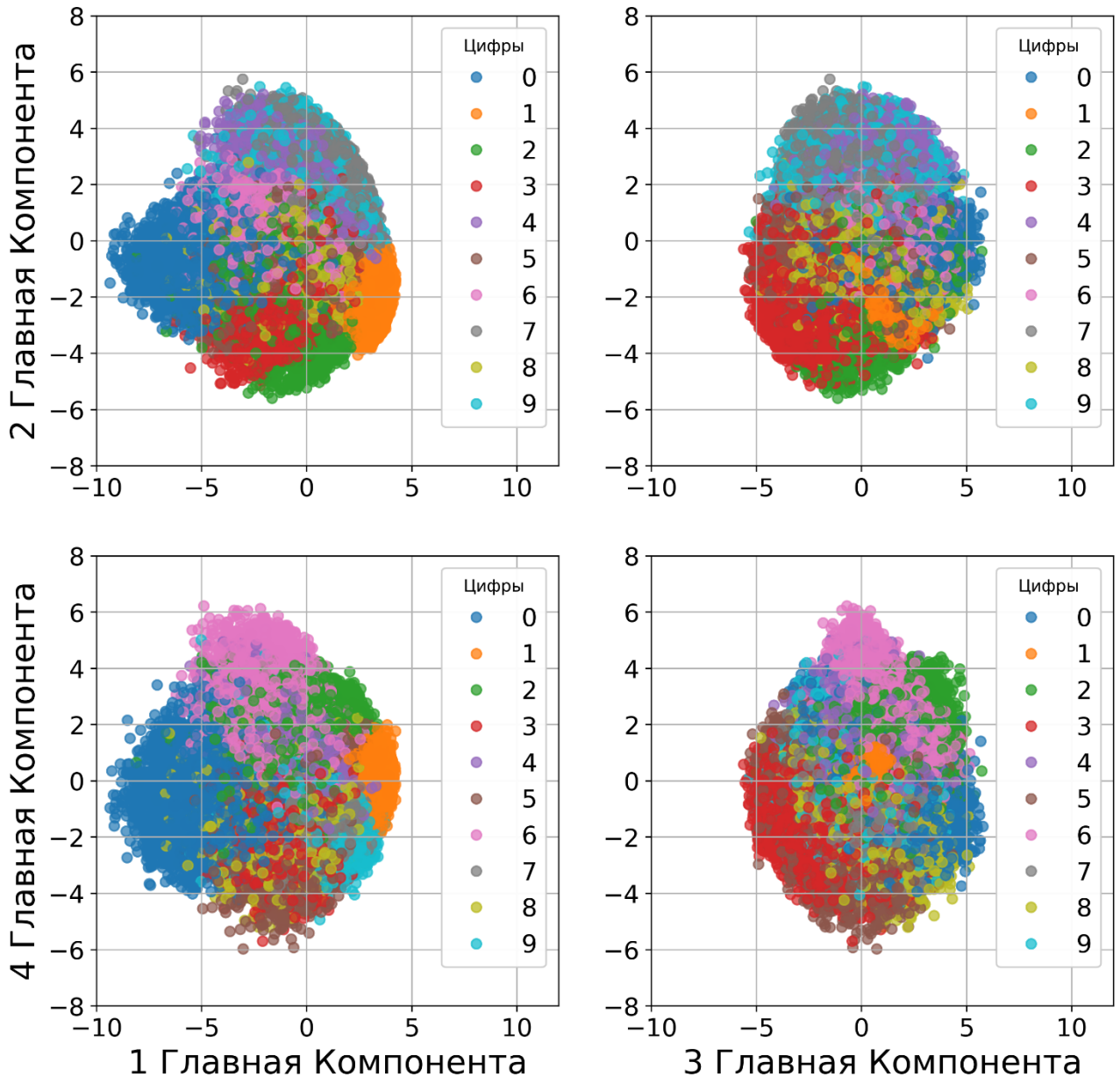


Рис. 7-4 Визуализация пространства 4 главных компонент для набора данных MNIST

Попробуем интерпретировать полученные результаты. Для этого визуализируем собственные вектора. Сгруппируем их таким образом, чтобы можно было составить изображение, аналогичное исходным. На Рис. 7-5 красным цветом обозначены положительные значения, а синим – отрицательные. С учетом того, что главные компоненты – это линейная комбинация исходных признаков, то «работу» первой главной компоненты можно интерпретировать следующим образом: берется центральная область

изображения с положительными весами, и окружающая ее область с отрицательными.

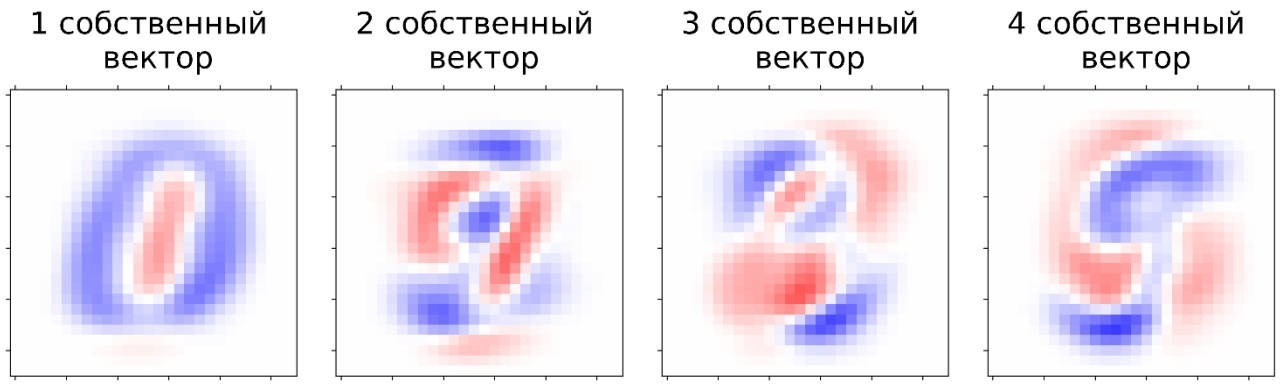


Рис. 7-5 Визуализация 4 собственных векторов для набора данных MNIST

Таким образом, если изображение похоже на цифру 1, то для этой главной компоненты получится положительное число (светлые пиксели исходного изображения будут помножены на положительные веса). Если изображение похоже на цифру 0, то для этой главной компоненты получится отрицательное число (светлые пиксели исходного изображения будут помножены на отрицательные веса). Для других цифр результат зависит от того сколько светлых пикселей попадут в зоны положительных и отрицательных весов.

Аналогичным образом можно попытаться интерпретировать другие главные компоненты. Однако, к сожалению, такой красивой и однозначной трактовки как с первой главной компонентой и цифрами 0 и 1 не получится.

Наконец посмотрим, как будут выглядеть изображения если их реконструировать из 100 главных компонент. На Рис. 7-6 представлены реконструированные изображения и соответствующие им оригиналы. Видно, что в целом, восстановленные изображения соответствуют оригиналам. При этом имеются шумы и артефакты.

Рис. 7-6 Реконструкция изображения с использованием главных компонент

Можно количественно оценить, коэффициент сжатия на уровне размерностей. Исходная матрица $n \times p$ восстанавливается из пространства главных компонент размером $n \times p^{'}$, и совокупности собственных векторов, общим размером $p \times p^{'}$. Тогда количественно коэффициент сжатия К можно оценить по следующей формуле

$$К = \frac{p^{'}*(p+n)}{p*n}$$

Для наших преобразований получим следующие значения

$$К = \frac{100*(784+70000)}{784*70000} \approx 0.13 = 13\%$$

Однако стоит отметить ограниченность метода главных компонент в общем случае. Так найденные собственные вектора будут относительно хорошо работать на новых данных, которые «похожи» на те, что были в обучающем наборе данных. Но если цифры будут наклонены под другим углом, то результаты восстановления могут быть неожиданными. Также если изображения более сложные, например, различные изображения разных котов, то собственные вектора могут носить случайных характер.

Ссылки на справочные материалы

1. Блог с описанием ключевых терминов и понятий машинного обучения «простыми словами». *Блок про уменьшение размерности (обобщение)* https://vas3k.ru/blog/machine_learning/
2. Плейлист с хорошей визуализацией понятий и концепций Линейной Алгебры (на английском). *Обратить внимание на видео про собственные значения и собственные вектора* https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab
3. Описание метода главных компонент https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B3%D0%BB%D0%B0%D0%B2%D0%BD%D1%8B%D1%85_%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82
4. Описание сингулярного разложение матриц https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%BE%D0%B5_%D1%80%D0%B0%D0%B7%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5

Примерные Вопросы для контроля

8. Напишите, как вы поняли, из-за чего происходит уменьшение размерности в методе главных компонент (вопрос-дискуссия)
9. Объясните, почему перед применением метода главных компонентов необходимо провести стандартизацию данных (вопрос-дискуссия)
10. Что означают собственные значения и собственные векторы ковариационной матрицы в методе главных компонент? (вопрос-дискуссия)
11. Как связаны главные компоненты с исходными данными?

12. Что означают матрицы U, S и V в сингулярном разложении?
13. В чем отличие между вариантами сингулярного разложения матриц: компактное, экономичное и усеченное?

8. Кластеризация

Список тем, которые должны быть осуждены на лекции:

1. Кластеризация
2. Расстояние
3. Метод к-Средних
4. Метрики Кластеризации

Ключевые моменты по темам:

1. Задача кластеризации. Обсуждение примеров задач из реальной жизни, которые можно свести к кластеризации.

   Области применения задач кластеризации (дискуссия)
2. Определение расстояния. Особенности различных метрик расстояния
   - Евклидово расстояние
   - Манхэттенское расстояние
   - Расстояние Чебышева
   - Расстояние Минковского

   Эквидистантные области при использовании разных метрик расстояния
3. Метод кластеризации к-Средних (k-Means)

   Описание ключевых шагов алгоритма кластеризации к-Средних

   Демонстрация работы алгоритма кластеризации к-Средних

   Метод локтя для определения оптимального числа к

   SWOT-анализ кластеризации методом к-Средних

   Обсуждение ситуаций, когда алгоритм к-Средних будет выдавать заведомо неправильные ответы
4. Обсуждение метрик кластеризации.

   Описание метода силуэтов для оценки кластеризации

Подведение Итогов Лекции

Текстовые материалы

Интересный исторический факт. Американский физик Стюарт Ллойд, выпускник знаменитой инновационной фабрики Bell Labs и Манхэттенского

проекта, в котором была изобретена атомная бомба, впервые предложил кластеризацию k-средних в 1957 году для распределения информации в цифровых сигналах. Он не публиковал его до 1982 года. Тем временем американский статистик Эдвард Форги описал аналогичный метод в 1965 году, что привело к его альтернативному названию — алгоритму Ллойда-Форги.

В ряде случаев при анализе данных оказывается так, что про данные ничего не известно, однако хочется понять на сколько они однородны или, например, могут быть разделены на группы. Такие группы можно назвать кластерами. Задача разделения на кластеры не требует наличия учителя. Другими словами, мы пытаемся найти закономерности в данных как таковых без привязки к тому, какие результаты для них мы хотим получить.

Метод кластеризации k-средних, как и многие другие методы кластеризации, опирается на «близость» точек данных друг к другу. Для этого нам необходимо научиться измерять расстояние между точками в произвольном пространстве.

Метрики расстояния

Для простоты описания используем двумерный случай, однако все описанные метрики будут работать и на случай пространств большей размерности.

Евклидово расстояние

Первая метрика расстояния знакома нам всем еще со времен школы, хотя не все смотрят на теорему Пифагора как способ определения расстояния между точками. Допустим у нас есть две точки в двумерном пространстве $a, b \in R^2$, координаты которых нам известны. Достроим прямоугольный треугольник (Рис. 8-1), тогда искомое расстояние будет гипотенузой в этом прямоугольном треугольнике.

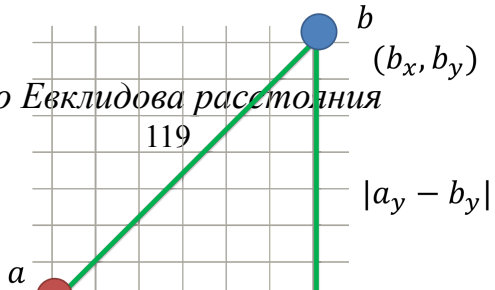


Рис. 8-1 К определению Евклидова расстояния

Согласно теореме Пифагора, квадрат гипотенузы равен сумме квадратов катетов. Катеты находятся как модуль разности соответствующих координат. Тогда искомое расстояние будет определено как корень квадратный из суммы квадратов разности координат. В общем случае, $a, b \epsilon R^m$ и расстояние определится по формуле $L^2 = d_2(a,b) = \{\sum_{i=1}^{m} |a_i - b_i|^2\}^{1/2}$. Эта метрика расстояния носит название **Евклидово расстояние**.

Манхэттенское расстояние

А теперь давайте представим ситуацию, что мы находимся в городе, и мы едем по машине (Рис. 8-2). Теперь, мы не можем двигаться напрямую по гипотенузе, как птицы, а вынуждены использовать дороги, перекрестки и т.д. Тогда расстояние будет определятся просто как сумма длин катетов.

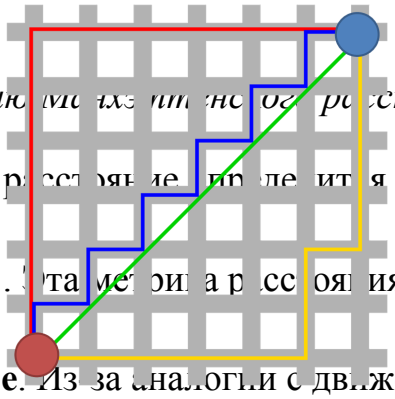


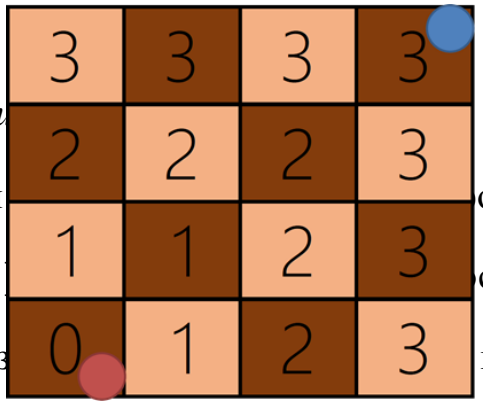
Рис. 8-2 К определению манхэттенского расстояния

В общем случае, $a, b \epsilon R^m$ и расстояние определится по формуле $L^1 = d_1(a,b) = \sum_{i=1}^{m} |a_i - b_i|$. Эта метрика расстояния носит название **Манхэттенское расстояние**. Из-за аналогии с движением по городу часто упоминаются названия «City Block» и «Taxicab».

Расстояние Чебышева

Интуитивным примером для следующей метрики расстояния служит шахматная доска и фигура король. Из правил шахмат мы помним, что король может двигаться на одну клетку в любом направлении: по горизонтали, по вертикали и диагонали. Исходя из этого можно оценить каждую клетку на доске с т. з. того, сколько ходов понадобиться королю чтобы дойти до этой точки (Рис. 8-3).

Рис. 8-3 К определени



В общем случае, $a, b \in R^m$ и ормуле

$L^\infty = d_\infty(a,b) = (|a_i - b_i|$ сит название

Расстояние Чебышева. Из городу часто упоминаются название «ChessBoard».

Расстояние Минковского

Нетрудно заметить, что общую составляющую в метриках расстояния. Используется модуль разности координат, полученные модули разности определенным образом суммируются: либо возводятся в квадрат (вторая степень), либо берутся просто модули (первая степень). Естественным образом метрика расстояния обобщается на произвольную степень $p \geq 1$. В общем случае, $a, b \in R^m$ и расстояние определится по формуле

$L^p = d_p(a,b) = \{\sum_{i=1}^{m} |a_i - b_i|^p\}^{1/p}$. Эта метрика расстояния носит название

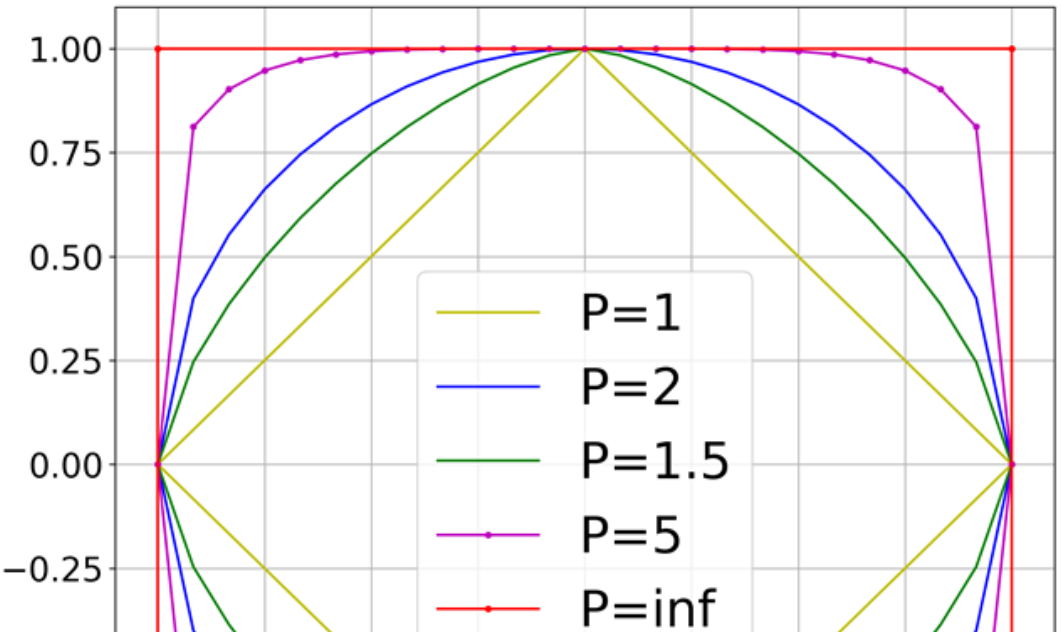
Расстояние Минковского.

На Рис. 8-4 представлены эквидистантные фигуры для различных метрик расстояний. Все точки, лежащие на прямых одного цвета, находятся на расстоянии 1 от центра координат с «точки зрения» соответствующей метрики расстояния.

P

Запише

```
def distar
if metr
dist
```



```
    if metric == 'cityblock':
        dist = np.sum(np.abs(X1 - X2).T,axis=0)
    if metric == 'Chebyshev':
        dist = np.max(np.abs(X1 - X2).T,axis=0)
    if metric == 'Minkowski':
        dist = np.power(np.sum(np.power(np.abs(X1 - X2),p).T,axis=0),1/p)
    return dist
```

Функция работает как с векторами, так и матрицами равной размерности.

Как правило, метрикой расстояния по умолчанию является Евклидова метрика. Однако в разных задачах, особенно если анализируются категориальные переменные могут хорошо проявлять себя и другие метрики. Выбор метрики расстояния тоже можно отнести к гиперпараметру модели.

Также необходимо не забывать о стандартизации или нормализации данных при использовании метрик расстояния. Иначе результаты вычисления метрик могут быть не корректны, если используются данные, измеряющиеся в разных диапазонах.

Алгоритм к-Средних

Одним из самых простых методов кластеризации является метод k-средних. Суть данного метода сводится к тому, чтобы найти заданное число кластеров (k) и их центры (т. н. центроиды) таких, чтобы расстояние от центроидов до всех точек кластера было минимальным.

Алгоритм k-средних может быть описан следующим образом

- выбирается к случайные точки - центроиды.
- рассчитывается вектор расстояние между каждой точкой набора данных и каждым центроидом.
- в каждый кластер записываются те точки для которых оказалось, что для соответствующего центроида расстояние меньше, чем для других.

- новые значение центроидов рассчитываются как среднее значение по всем точкам кластера.

В качестве данных рассмотрим генерируемые данные из главы 4. Начнем с простой модели линейно разделимых данных.

Прежде чем проводить кластеризацию необходимо проинициализировать кластеры. Для этого выберем случайные индексы среди доступных в наборе данных.

```
def init_centroids(X, n_clusters):
    centroid_idxs = np.random.randint(0, X.shape[0], size = n_clusters)
    return X[centroid_idxs,:]
```

Посмотрим, как это работает для двух кластеров. Проведем первую кластеризацию. Для этого возьмем каждый центроид и посчитаем расстояние от него до все записей набора данных. Индексы значений для каждого кластера выберем как индексы минимальных расстояний до соответствующего центроида. Таким образом, нулевой кластер будет включать те точки набора данных, в которых расстояние до нулевого центроида меньше, чем до первого центроида.

```
def predict(X, n_clusters, centroids, metric = 'euclidean', p = 2):
    distances = np.zeros((X.shape[0], n_clusters))
    for i,centr in enumerate(centroids):
        distances[:,i] = distance(centr,X, metric, p)
    cluster_label = np.argmin(distances,axis = 1)
    return cluster_label
```

Посмотрим, как распределились результаты кластеризации после случайной инициализации центроидов (Рис. 8-5). Получилось достаточно интересно:

оба центроида оказались относительно близко друг к другу. Тем не менее, это нулевая итерация алгоритма.

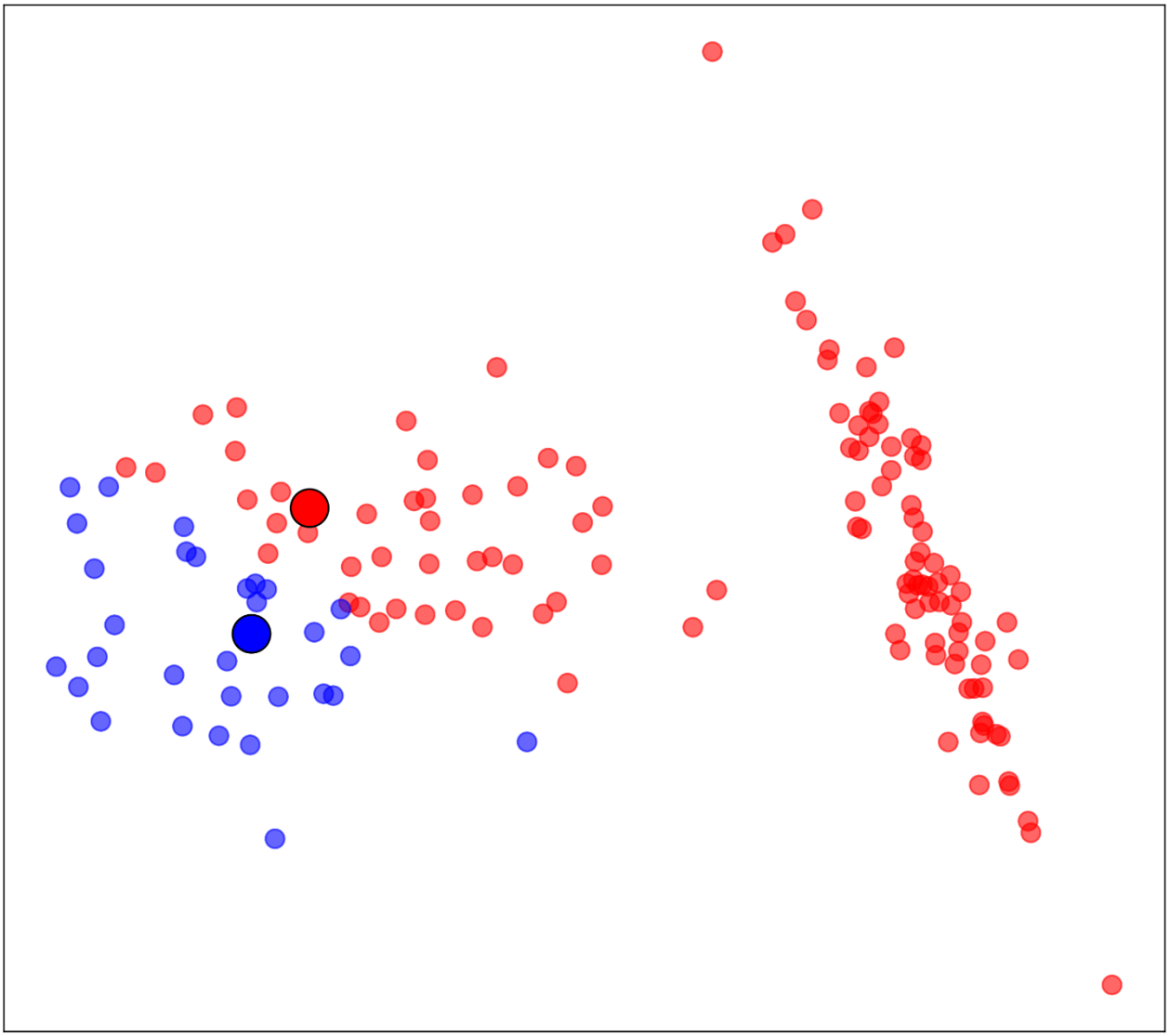


Рис. 8-5 Результаты Кластеризации после Инициализации центроидов

Теперь выберем новые центроиды. Новый центроид для каждого кластера выберем как среднее значение по кластеру. В визуализации вы увидите старый центроид как звезду и новый как фиолетовый круг (Рис. 8-6). Видно, что центр для «красного» кластера значительно сместился.

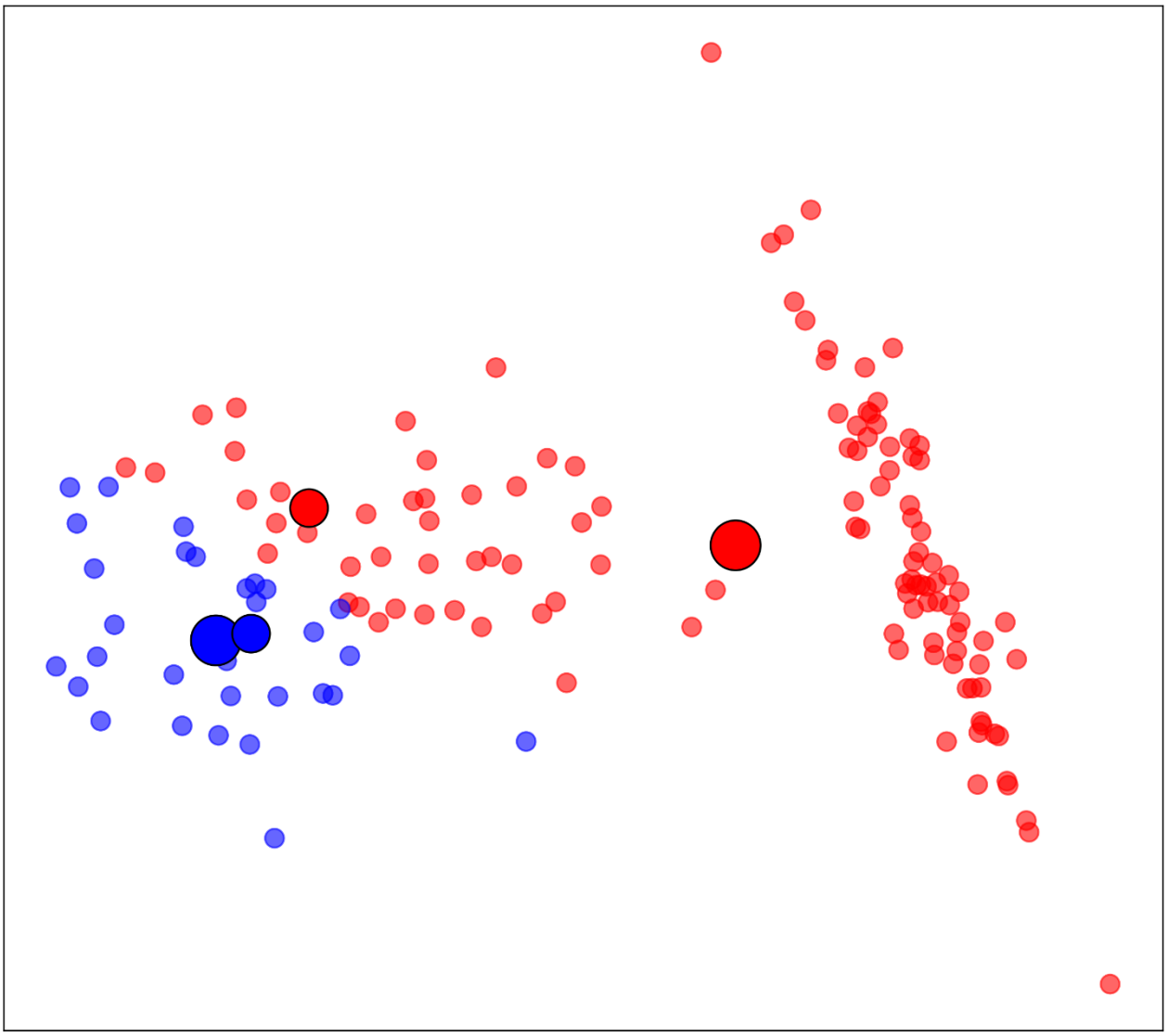


Рис. 8-6 Результаты Кластеризации после Изменения центроидов

Рассчитаем относительное расстояние между старыми и новыми центроидами. Если расстояние между обновленными центроидами будет сравнительно небольшим - то есть центроиды перестанут менять позицию, то мы будем считать, что кластеризация закончена.

```
def delta_centroids(centroids,old_centroids, metric = 'euclidean', p = 2):
    return (distance(centroids,old_centroids, metric, p)/distance(old_centroids, np.mean(old_centroids), metric, p)).mean()
```

Для автоматизации процесса реализуем процедуру итерационной кластеризации. Укажем порог относительного изменения центроидов - tol. В конце процедуры выведем результирующий номер итерации и расстояние между последним изменением центроидов

```
def fit(X, n_clusters, centroids, max_iter=10, tol=0.01, metric = 'euclidean', p = 2):


  dcentr = np.inf


  for i in range(max_iter):


    old_centroids = np.copy(centroids)

    cluster_label=predict(X, n_clusters, centroids, metric, p)


    for k in range(n_clusters):

      c_idxs = np.flatnonzero(cluster_label==k)

      centroids[k] = X[c_idxs].mean(axis = 0)


    dcentr = delta_centroids(centroids,old_centroids, metric, p)


    if dcentr<=tol:

      break


  print('Мы остановились на итерации:', i,', относительное изменение центроидов: ',dcentr)


  return cluster_label
```

Проверим и визуализируем результаты. Буквально через пару итераций мы получим следующее распределение (Рис. 8-7).

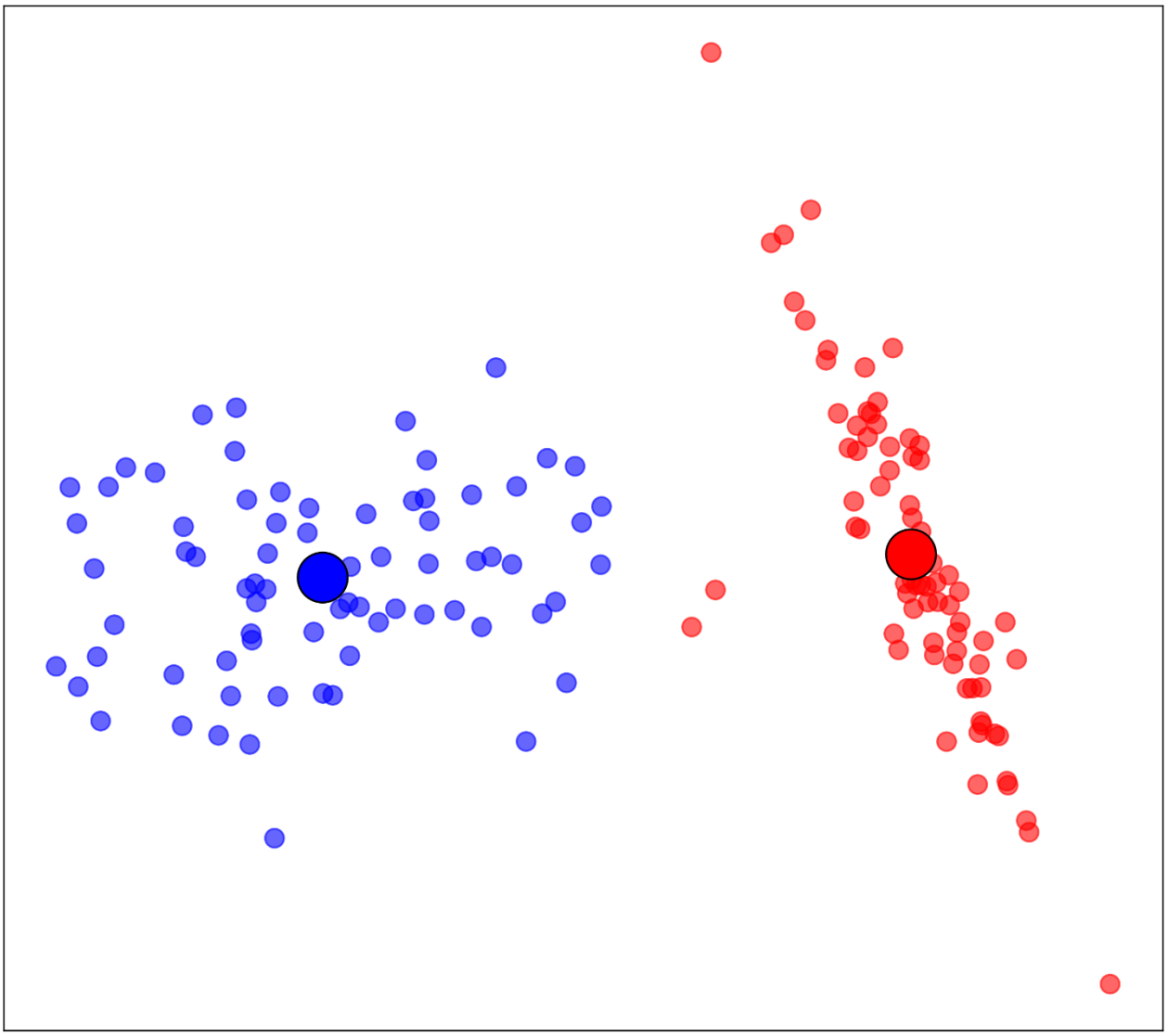


Рис. 8-7 Результаты Кластеризации после нескольких итераций

Теперь объединим все наши наработки в один класс KMeans, который представлен в Приложении 4. Центроиды на последней итерации хранятся в атрибуте .centroids. Для того чтобы получить предсказания номера кластера необходимо воспользоваться методом .fit_transform. Также мы добавили в класс KMeans атрибут **инерция** .inertia – сумму квадратов расстояния точек кластеров до соответствующих центроидов.

С помощью этого параметре можно попытаться ответить на вопрос, а какое число кластеров оптимальное для конкретных данных. Для этого можно

использовать т. н. **метод Локтя** (Elbow Method). проверяются различные количества кластеров и запоминаются конечные значения инерции для каждого числа кластеров. Затем визуализируется зависимость (число кластеров, инерция). Как правило получаются зависимости, подобные Рис. 8-8.

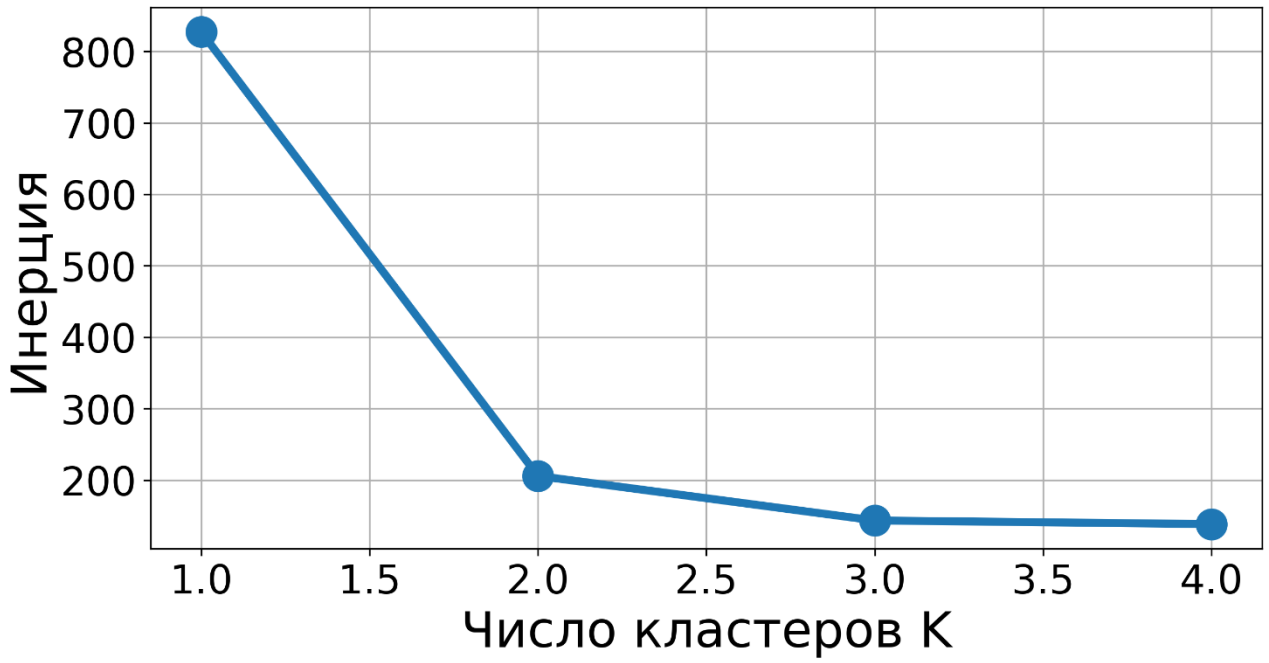


Рис. 8-8 Визуализация Метода Локтя

Затем ищется такое число кластеров, которое напоминает перегиб согнутого локтя (в нашем случае это число к = 2). Большее количество кластеров не уместно – мы начинаем искусственно дробить большие кластеры, на малые, нарушая целостную структуру данных (Рис. 8-9).

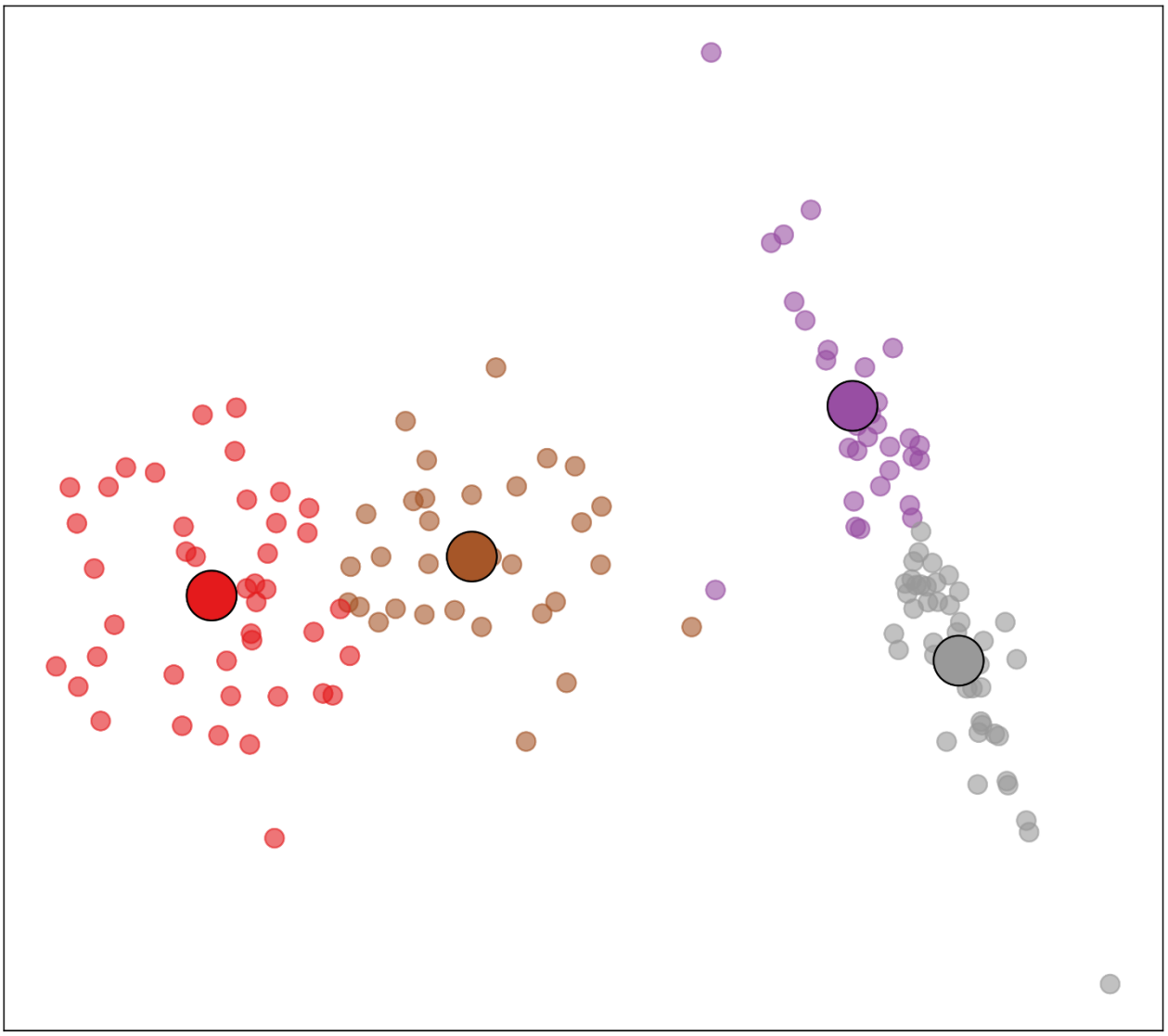


Рис. 8-9 Кластеризация К-Средних с слишком большим числом кластеров

Метод К-Средних можно применять не только к двумерным данным, но и к данным большей размерности. При этом рекомендуется пользоваться методами уменьшения размерности для возможности визуализации результата кластеризации.

В общем случае кластеризация применяется в контексте того, что истинные метки кластеров нам не известны. Именно поэтому эту задачу машинного обучения относят к задачам «обучения без учителя». Однако иногда возникают тренировочные задачи, в которых метки известны. Или помимо числовых данных у нас есть набор категориальных признаков, и мы хотим

выяснить, связаны ли найденные кластеры с каким-то категориальным признаком.

Для этого можно воспользоваться схожей матрицей с матрицей ошибок из главы 4 и соответствующих метрик. Только нужно помнить особенность – номера кластеров генерируются случайно, и могут совершенно противоречить известным меткам классов. Проверить это можно с помощью перекрестного табулирования с использованием метода crosstab библиотеки Panda: здесь y – истинные метки классов, а c_labels – полученные предсказания кластеров.

```
pd.crosstab(y,c_labels, rownames=['Метки'], colnames = ['Предсказания'])
```

И уже на основе таблицы кросс-табулирования можно оценивать метрики схожие с метриками классификации, введя следующие обозначения

- **TP** элементы принадлежат одному кластеру и одному классу
- **FP** элементы принадлежат одному кластеру, но разным классам
- **FN** элементы принадлежат разным кластерам, но одному классу
- **TN** элементы принадлежат разным кластерам и разным классам

- Индекс Rand (аналог доли правильных ответов) $Rand = \frac{TP+TN}{TP+TN+FP+FN}$
- Индекс Жаккара $Jaccard = \frac{TP}{TP+FP+FN}$
- Индекс Фоулкса – Мэллова $FM = \sqrt{\frac{TP}{TP+FP}\frac{TP}{TP+FN}}$

Ссылки на справочные материалы

1. Блог с описанием ключевых терминов и понятий машинного обучения «простыми словами». *Блок про кластеризацию* https://vas3k.ru/blog/machine_learning/

2. Описание кластеризации методом к-средних https://en.wikipedia.org/wiki/K-means_clustering
3. Описание метрик кластеризации https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%86%D0%B5%D0%BD%D0%BA%D0%B0_%D0%BA%D0%B0%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%B0_%D0%B2_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B5_%D0%BA%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8

Примерные Вопросы для контроля

1. Какая связь между евклидовым расстоянием и расстоянием Минковского? (вопрос-дискуссия)
2. Что является основным гиперпараметром алгоритма кластеризации k-средних? (к – число кластеров)
3. Может ли коэффициент силуэта быть равным отрицательному числу? Если «да» - в каких случаях, если «нет» - почему? (Может. Если «свой кластер» занимает большое место, а ближайший чужой – компактный и близкий)
4. Как можно использовать метод локтя для определения оптимального количества кластеров? (вопрос-дискуссия)

9. **Библиотека scikit-learn**

Список тем, которые должны быть осуждены на лекции:

1. Предварительна Обработка
2. Разложение Матриц
3. Кластеризация
4. Линейная Регрессия
5. Логистическая Регрессия

Ключевые моменты по темам:

1. Пакет sklearn.preprocessing для предварительной обработки данных
   Общая структура использования методов .fit(), .transform(), .inverse_transfrorm()
   Стандартизация (StandardScaler), нормализация (MinMaxScaler), Степенное преобразование (PowerTransformer)
2. Пакет sklearn.decomposition для разложения матриц
   Метод главных компонент (PCA)
3. Пакет sklearn.cluster для кластеризации
   Кластеризация к-Средних (KMeans). Общая структура использования методов .fit(), .predict()
   Метрики кластеризации
4. Пакет sklearn.linear_model для линейных моделей
   Линейная регрессия (LinearRegression). Объект полиномиальные признаки (PolynomialFeatures)
   Пакет sklearn.pipeline
   Регуляризация линейной регрессии (Lasso и Ridge)
   Метрики регрессии
   Модуль sklearn.model_selection для выбора моделей.
   Перекрестная проверка (cross_validate, ShuffleSplit)
5. Логистическая регрессия (LogisticRegression)
   Метрики классификации

Перекрестная проверка (cross_validate, StratifiedKFold)

Подведение Итогов Лекции

Ссылки на справочные материалы

1. Документация библиотеки scikit-learn для предварительной обработки данных https://scikit-learn.org/stable/modules/preprocessing.html
2. Документация библиотеки scikit-learn для метода главных компонент https://scikit-learn.org/stable/modules/decomposition.html#pca
3. Документация библиотеки scikit-learn для кластеризации к-средних https://scikit-learn.org/stable/modules/clustering.html#k-means
4. Документация библиотеки scikit-learn для линейных моделей https://scikit-learn.org/stable/modules/linear_model.html
5. Документация библиотеки scikit-learn для перекрестной проверки https://scikit-learn.org/stable/modules/cross_validation.html
6. Документация библиотеки scikit-learn для составных моделей https://scikit-learn.org/stable/modules/compose.html

10.Продвинутые алгоритмы кластеризации

Список тем, которые должны быть осуждены на лекции:

1. Иерархическая Кластеризация
2. Кластеризация DBSCAN

Ключевые моменты по темам:

1. Иерархическая Кластеризация

   Описание ключевых шагов алгоритма Иерархической Кластеризации

   Демонстрация работы алгоритма Иерархической Кластеризации

   Типы связей (linkage)

   Дендрограмма

   Реализация scikit-learn (пакет sklearn.cluster – AgglomerativeClustering)

   SWOT-анализ Иерархической Кластеризации
2. Метод кластеризации DBSCAN

   Описание ключевых шагов алгоритма кластеризации DBSCAN

   Демонстрация работы алгоритма кластеризации DBSCAN

   Реализация scikit-learn (пакет sklearn.cluster – DBSCAN)

   SWOT-анализ кластеризации DBSCAN

Подведение Итогов Лекции

Ссылки на справочные материалы

1. Документация библиотеки scikit-learn для иерархической кластеризации
   https://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering
2. Документация библиотеки scikit-learn для кластеризации DBSCAN
   https://scikit-learn.org/stable/modules/clustering.html#dbscan

Примерные Вопросы для контроля

1. Иерархическая кластеризация: в чем разница между разными типами связей? (вопрос-дискуссия)
2. Иерархическая кластеризация: какую метрику вы используете при выборе количества кластеров по дендрограмме? (вопрос-дискуссия)
3. Иерархическая кластеризация: в чем разница между разными типами связей? (вопрос-дискуссия)
4. DBSCAN: что может случиться, если вы установите слишком высокое значение epsilon? (вопрос-дискуссия)

Ближайшие Соседи

Список тем, которые должны быть осуждены на лекции:

1. Классификация методом к-Ближайших Соседей
2. Регрессия методом к-Ближайших Соседей
3. Neighborhood Components Analysis
4. t-SNE

Ключевые моменты по темам:

1. Алгоритм классификации методом к-Ближайших Соседей
   Влияние гиперпараметров метода на результат классификации (количество соседей, веса, расстояние)
   Реализация scikit-learn (пакет sklearn.neighbors – KNeighborsClassifier)
   Выбор оптимальных гиперпараметров (пакет sklearn.model_selection – GridSearchCV)
2. Алгоритм регрессии методом к-Ближайших Соседей
   Влияние гиперпараметров метода на результат регрессии (количество соседей, веса, расстояние)
   Реализация scikit-learn (пакет sklearn.neighbors – KNeighborsRegressor)
   SWOT-анализ метода к-Ближайших соседей
3. Общая идеология применения метода к-ближайших соседей, перебор оптимальных гиперпараметров
   Neighborhood Components Analysis: обучаемое линейное преобразование параметров для оптимизации метода к-ближайших соседей
   Реализация scikit-learn (пакет sklearn.neighbors – NeighborhoodComponentsAnalysis)
   SWOT-анализ Neighborhood Components Analysis
4. Визуализация данных методом t-SNE
   Подобие, нормированное подобие, t-распределение
   Применение градиентного спуска для построения нового пространства данных, дивергенция Кульбака-Лейблера

Perplexity

Реализация scikit-learn (пакет sklearn.manifold – TSNE)

SWOT-анализ метода t-SNE

Подведение Итогов Лекции

Ссылки на справочные материалы

1. Описание метода ближайших соседей https://ml-handbook.ru/chapters/metric_based/intro#%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-k-%D0%B1%D0%BB%D0%B8%D0%B6%D0%B0%D0%B9%D1%88%D0%B8%D1%85-%D1%81%D0%BE%D1%81%D0%B5%D0%B4%D0%B5%D0%B9-knn
2. Оригинальная статья про метод Neighborhood Components Analysis https://cs.nyu.edu/~roweis/papers/ncanips.pdf
3. Интерактивная визуализация метода t-SNE https://distill.pub/2016/misread-tsne/
4. Документация библиотеки scikit-learn для методов ближайших соседей https://scikit-learn.org/stable/modules/neighbors.html
5. Документация библиотеки scikit-learn для метода t-SNE https://scikit-learn.org/stable/modules/manifold.html#t-distributed-stochastic-neighbor-embedding-t-sne

Примерные Вопросы для контроля

1. В чем основное отличие использования метода k-ближайших соседей для задач классификации и регрессии? (вопрос-дискуссия)
2. Как найти оптимальное значение k для методов ближайших соседей? (вопрос-дискуссия)
3. Как можно уменьшить размерность данных с помощью Neighborhood Components Analysis? (вопрос-дискуссия)
4. Какой функционал оптимизируется при использовании градиентного спуска для Neighborhood Components Analysis? (вопрос-дискуссия)

5. Какой гиперпараметр в реализации t-SNE связан с балансом между локальными и глобальными аспектами структуры данных? (Perplexity)
6. Можно ли использовать обученное преобразование t-SNE на новых данных? (Нет)

11. Метод Опорных Векторов

Список тем, которые должны быть осуждены на лекции:

1. Метод Опорных Векторов
2. Kernel Trick
3. Регрессия методом опорных векторов

Ключевые моменты по темам:

1. Ключевые понятия метода опорных векторов: опорный вектор, гиперплоскость, зазор

   Метод опорных векторов с жестким зазором

   Применение Функции Лагранжа для поиска весовых коэффициентов метода опорных векторов

   Метод опорных векторов с мягким зазором, сачкующие (slack) точки
2. Ядерное преобразование (Kernel Trick)

   Достаточность знания ядра для метода опорных векторов

   Разбор примеров ядер: полиномиальное ядро, радиальное ядро

   Применение метода опорных векторов для многоклассовой классификации

   Реализация scikit-learn (пакет sklearn.svm – SVC)
3. Метод опорных векторов для задачи регрессии

   Особенности применения метода для задачи регрессии

   SWOT-анализ метода опорных векторов

Подведение Итогов Лекции

Ссылки на справочные материалы

1. Описание метода опорных векторов
   https://ml-handbook.ru/chapters/linear_models/intro#hinge-loss-svm
2. Документация библиотеки scikit-learn для метода опорных векторов
   https://scikit-learn.org/stable/modules/svm.html

Примерные Вопросы для контроля

1. Какие точки считаются опорными векторами (для задач классификации и регрессии)? (вопрос-дискуссия)
2. В чем разница между Hard Margin SVM (жестким зазором) и Soft Margin SVM (мягким зазором)? (вопрос-дискуссия)
3. Почему Kernel trick помогает улучшить результаты метода опорных векторов? (вопрос-дискуссия)
4. Почему для Kernel trick достаточно знать только ядро преобразования, а не само преобразование? (вопрос-дискуссия)

12.Байесовские Методы

Список тем, которые должны быть осуждены на лекции:

1. Теорема Байеса
2. Наивный Байесовский классификатор
3. Дискриминантный анализ

Ключевые моменты по темам:

1. Теорема Байеса

   Гипотеза и Событие; Апостериорная вероятность, априорная вероятность, правдоподобие

   Разбор теоремы Байеса на примере
2. Наивный Байесовский классификатор

   «Наивное» предположение о независимости признаков

   Разбор применения Байесовского классификатора на примере

   Использование различных распределений при работе с разными типами данных: категориальное распределение, нормальное распределение, распределение Бернулли

   Реализация scikit-learn (пакет sklearn.naive_bayes – CategoricalNB, GaussianNB, BernoulliNB)

   SWOT-анализ Наивного Байесовского классификатора
3. Дискриминантный анализ

   Многомерное нормальное распределение

   Линейный дискриминант Фишера

   Линейный дискриминатный анализ (вероятностный подход)

   Квадратичный дискриминантный анализ

   Реализация scikit-learn (пакет sklearn.discriminant_analysis – LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis)

   SWOT-анализ дискриминантного анализа

Подведение Итогов Лекции

Ссылки на справочные материалы

4. Документация библиотеки scikit-learn для наивного байесовского классификатора https://scikit-learn.org/stable/modules/naive_bayes.html
5. Документация библиотеки scikit-learn для дискриминантного анализа https://scikit-learn.org/stable/modules/lda_qda.html

Примерные Вопросы для контроля

1. Что означает «наивный» в наивном байесовском классификаторе? (вопрос-дискуссия)
2. В чем разница между линейным и квадратичным дискриминантным анализом? (вопрос-дискуссия)
3. В чем разница между гауссовским наивным байесовским классификатором и дискриминантным анализом? (вопрос-дискуссия)
4. Почему линейный дискриминантный анализ может использоваться как метод уменьшения размерности? (вопрос-дискуссия)

13.Деревья Решений

Список тем, которые должны быть осуждены на лекции:

1. Деревья Решений для классификации и регрессии

Ключевые моменты по темам:

1. Деревья Решений

   Корневой узел, узлы принятия решений, листья

   Особенности применения деревьев решений для классификации и регрессии

   Критерий Джини, Прирост Информации, MSE

   Оценка значимости признаков с помощью деревьев решений

   Реализация scikit-learn (пакет sklearn.tree – DecisionTreeRegressor, DecisionTreeClassifier)

   SWOT-анализ Деревьев решений

Подведение Итогов Лекции

Ссылки на справочные материалы

1. Документация библиотеки scikit-learn для деревьев решений https://scikit-learn.org/stable/modules/tree.html#tree

Примерные Вопросы для контроля

1. В чем разница между использованием деревьев решений в задачах классификации и регрессии? (вопрос-дискуссия)
2. Какие типы данных могут использоваться деревьями решений? (вопрос-дискуссия)
3. Как выбираются наиболее оптимальные узлы решения? (вопрос-дискуссия)

14.Ансамблевые методы

Список тем, которые должны быть осуждены на лекции:

1. Методы бустинга
2. Методы усреднения

Ключевые моменты по темам:

1. Методы Бустинга

   AdaBoost

   Слабые модели, Обновление весов при ошибках

   Реализация scikit-learn (пакет sklearn.ensemble– AdaBoostRegressor, AdaBoostClassifier)

   SWOT-анализ AdaBoost

   Gradient Boosting

   Слабые модели, Прогноз остатков, Функция ошибки

   Реализация scikit-learn (пакет sklearn.ensemble– GradientBoostingRegressor, GradientBoostingClassifier)

   Реализация XGBoost

   Реализация LightGBM

   Реализация LightGBM

   SWOT-анализ Gradient Boosting
2. Методы усреднения

   Классификатор голосование

   Bagging, Bootstrapping

   Алгоритм Случайный лес

   Реализация scikit-learn (пакет sklearn.ensemble– RandomForestRegressor, RandomForestClassifier)

Подведение Итогов Лекции

Ссылки на справочные материалы

1. Документация библиотеки scikit-learn для ансамблевых методов https://scikit-learn.org/stable/modules/ensemble.html

2. Документация библиотеки XGBoost https://xgboost.readthedocs.io/en/stable/
3. Документация библиотеки LightGBM https://lightgbm.readthedocs.io/en/latest/
4. Документация библиотеки CatBoost https://catboost.ai

Примерные Вопросы для контроля

1. В чем разница между методами бустинга и методами усреднения? (вопрос-дискуссия)
2. Что такое «слабый предсказатель» в контексте ансамблевых методов? (вопрос-дискуссия)