МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

Институт математики и компьютерных технологий

Департамент информационных и компьютерных систем

Дмитрий Александрович Оськин

ПРАКТИЧЕСКИЙ АНАЛИЗ ДАННЫХ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON

Рекомендовано методическим советом ДВФУ в качестве учебного пособия для студентов, обучающихся по направлениям подготовки 09.03.03, 09.04.03 «Прикладная информатика»

Владивосток
2023

УДК 004.432

Рецензенты:

канд. техн. наук В. Е. Маркин (Морской государственный университет им. адм. Г. И. Невельского, кафедра автоматических и информационных систем);

доц., канд. техн. наук Я. В. Пафнутьева (Морской государственный университет им. адм. Г. И. Невельского, кафедра вычислительной техники)

Оськин Д. А.

Практический анализ данных на языке программирования python: учебное пособие / Д. А. Оськин. – Владивосток: Изд-во ДВФУ, 2023. – 141с.

Пособие содержит сведения об обработке и практическом анализе данных на языке программировании Python и является основой для изучения курсов «Теоретические основы и технология обработки больших данных», «Системы искусственного интеллекта». Предложен новый подход к использованию аналитики данных для исследования временных рядов и наборов данных.

Может быть рекомендовано студентам различных специальностей вузов, занимающихся разработкой систем искусственного интеллекта, математическим моделированием, статистическим и эконометрическим анализом Больших Данных, визуализацией результатов с применением пакетов языка программирования python – numpy, pandas, matplotlib, scikit-learn.

Библиогр.: 8 назв.

СОДЕРЖАНИЕ

Python для анализа данных

Python появился еще в 1990 году, но начал приобретать популярность не так давно. В 2020 Python стал четвертым в списке самых используемых языков программирования после JavaScript и SQL — его используют 44,1% разработчиков.

Python — это интерпретируемый, высокоуровневый объектно-ориентированный язык общего назначения, используемый для разработки API, искусственного интеллекта, веб-разработки, интернета вещей и так далее.

Отчасти Python стал так популярен благодаря специалистам в области datascience. Это один из самых простых языков для изучения. Он предлагает множество библиотек, которые применяются на всех этапах анализа данных. Поэтому язык однозначно подходит для этих целей.

Этапы анализа данных с Python

Python отлично работает на всех этапах. В первую очередь в этом помогают различные библиотеки. Поиск, обработка, моделирование (вместе с визуализацией) — 3 самых популярных сценария использования языка для анализа данных.

Поиск данных

Инженеры используют веб-краулингидля поиска данных с помощью Python.С их помощью можно создавать программы, которые собирают структурированные данные в сети. Также они собирают данные и расставляют их в определенном формате.

Обработка и моделирование данных

На этом этапе в числе самых используемых библиотек NumPy,Pandas и Scikit-learn (sklearn).

NumPy (NumericalPython) используется для сортировки больших наборов данных. Он упрощает математические операции и их векторизацию на массивах.

Pandas предлагает два структуры данных: Series (список элементов) и DataFrames (таблица с несколькими колонками). Эта библиотека конвертирует данные в DataFrame, позволяя удалять и добавлять новые колонки, а также выполнять разные операции.

Scikit-learn - один из наиболее широко используемых пакетов Python для DataScience и MachineLearning.

Визуализация данных

Matplotlib и Seaborn широко используются для визуализации данных. Они помогают конвертировать огромные списки чисел в удобные графики, гистограммы, диаграммы, тепловые карты и так далее.

Конечно, библиотек куда больше. Python предлагает бесчисленное количество инструментов для проектов в сфере анализа данных и может помочь при выполнении любых задач в процессе.

Преимущества и недостатки Python для анализа данных

Почти невозможно найти идеальный язык для анализа данных, поскольку у каждого есть свои достоинства и недостатки. Один лучше подходит для визуализации, а другой лучше работает с большими объемами данных. Выбор зависит и от личных предпочтений разработчика. Посмотрим на преимущества и недостатки Python для анализа данных.

Преимущества Python

Программирование никогда не было простым, и даже разработчики с большим количеством опыта сталкиваются с проблемами. К счастью, у каждого языка есть

сообщество, помогающее находить верные решения. На GitHub, например, более 90000 репозиториев с Python-проектами. Поэтому почти всегда можно найти ответ на интересующий вопрос.

Python — один из самых простых языков для изучения благодаря его простому синтаксису и читаемости. Он также требует куда меньшего количества строк кода. Разработчик может не думать о самом коде, а о том, что тот делает. Заниматься отладкой на Python тоже намного проще.

Python используется в самых разных отраслях благодаря его гибкости и широкому набору инструментов.

Для Python существует огромное количество библиотек, которые можно использовать на разных этапах анализа данных. Плюс, большая часть из них — бесплатные. Это все влияет на простоту работы с данными с помощью Python.

Недостатки

Python — язык общего назначения и был создан не только для анализа данных. Разрабатывать с динамической типизацией куда проще, однако это замедляет поиск ошибок в данных, связанных с разными типами.

Альтернативы Python для анализа данных

Хотя Python и считается одним из главных языков для анализа данных, существуют и другие варианты. Каждый из таких языков предназначен для выполнения конкретной задачи (поиска данных, визуализации или работы с большими объемами данных), а некоторые и вовсе были разработаны специально для анализа данных и статистических вычислений.

R

R — второй по популярности язык для анализа данных, который часто сравнивают с Python. Он был разработан для статистических вычислений и графики, что отлично подходит для анализа данных. В нем есть инструменты для визуализации данных. Он совместим с любыми статистическими приложениями, работает офлайн, а разработчикам предлагаются различные пакеты для управления данными и создания графиков.

SQL

Широко используемый язык для запросов данных и редактирования. Это также отличный инструмент для хранения и получения данных. SQL прекрасно работает с большими базами данных и способен получать данные из сети быстрее остальных языков.

Julia

Julia был разработан для datascience и научных вычислений. Это относительно новый язык, который быстро приобретает популярность среди специалистов в области. Главная его цель — преодолеть недостатки Python и стать выбором №1 среди инженеров. Julia — компилируемый язык, что подразумевает более высокую производительность. Однако синтаксис похож на Python, пусть и с акцентом на математику. В Julia можно использовать библиотеки из Python, C и Forton. Также язык славится параллельными вычислениями, которые работают быстрее и сложнее чем в Python.

Процесс анализа данных

Когда проблема определена и задокументирована, можно двигаться к этапу планирования проекта анализа данных. Планирование необходимо для понимания того, какие профессионалы и ресурсы понадобятся для выполнения требований проекта максимально эффективно. Таким образом задача — рассмотреть те вопросы в области, которые касаются решения этой проблемы.

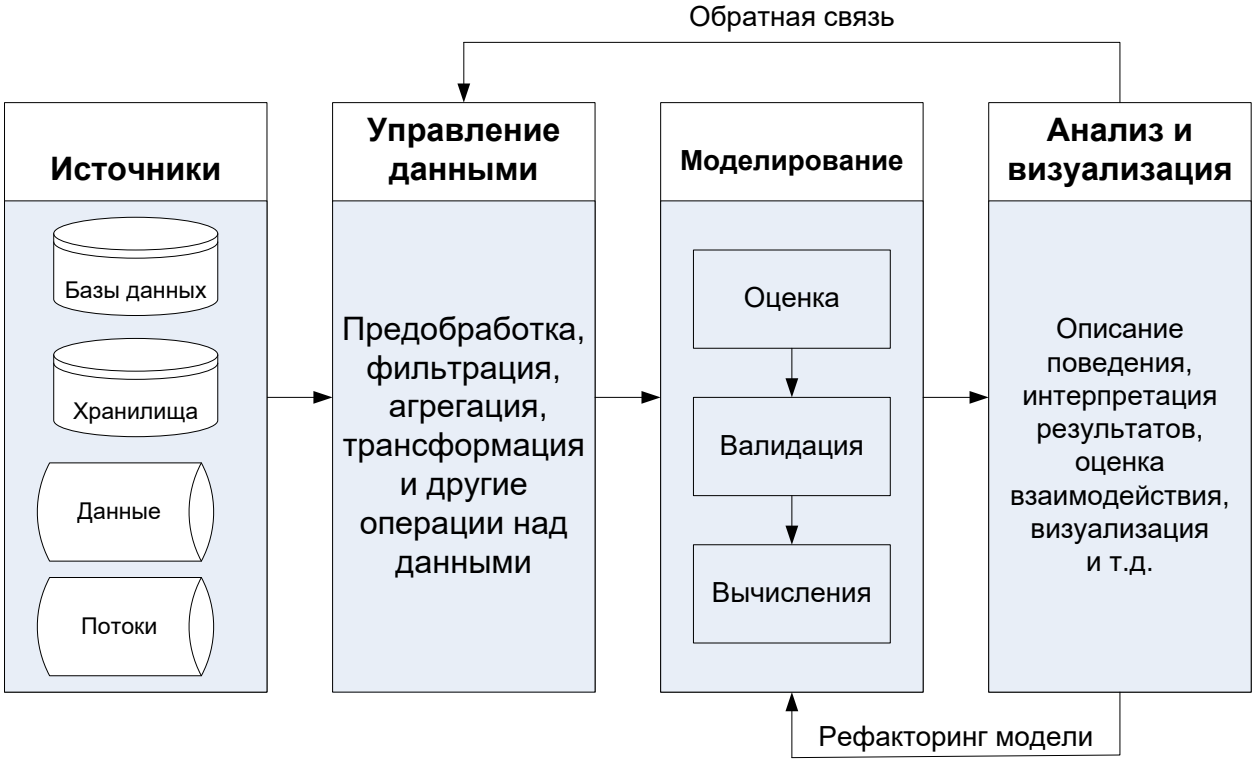


Рис. 1. Структура системы обработки Больших данных

Извлечение данных

Когда проблема определена, первый шаг для проведения анализа — получение данных. Они должны быть выбраны с одной базовой целью — построение предсказательной модели. Поэтому выбор данных — также важный момент для успешного анализа.

Данные должны максимально отражать реальный мир — то, как система реагирует на него. Например, использовании больших наборов сырых данных, которые были собраны неграмотно, это привести либо к неудаче, либо к неопределенности.

Поэтому недостаточное внимание, уделенное выбору данных или выбор таких, которые не представляют систему, приведет к тому, что модели не будут соответствовать изучаемым системам.

Поиск и извлечение данных часто требует интуиции, границы которой лежат за пределами технических исследований и извлечения данных. Этот процесс также требует понимания природы и формы данных, предоставить которое может только опыт и знания практической области проблемы.

Вне зависимости от количества и качества необходимых данных важный вопрос — использование лучших источников данных.

Если средой изучения выступает лаборатория (техническая или научная), а сгенерированные данные экспериментальные, то источник данных легко определить. В этом случае речь идет исключительно о самих экспериментах.

Но при анализе данных невозможно воспроизводить системы, в которых данные собираются исключительно экспериментальным путем, во всех областях применения.

Многие области требуют поиска данных в окружающем мире, часто полагаясь на внешние экспериментальные данные или даже на сбор их с помощью интервью и опросов.

В таких случаях поиск хорошего источника данных, способного предоставить все необходимые данные, — задача не из легких. Часто необходимо получать данные из нескольких источников данных для устранения недостатков, выявления расхождений и с целью сделать данные максимально общими.

Интернет — хорошее место для начала поиска данных. Но большую часть из них не так просто взять. Не все данные хранятся в виде файла или базы данных. Они могут содержаться в файле `HTML` или другом формате. Тут на помощь приходит техника парсинга. Он позволяет собирать данные с помощью поиска определенных HTML-тегов на страницах. При появлении таких совпадений специальный софт извлекает нужные данные. Когда поиск завершен, у вас есть список данных, которые необходимо проанализировать.

Подготовка данных

Из всех этапов анализа подготовка данных кажется наименее проблемным шагом, но на самом деле требует наибольшего количества ресурсов и времени для завершения. Данные часто собираются из разных источников, каждый из которых может предлагать их в собственном виде или формате. Их нужно подготовить для процесса анализа.

Подготовка данных включает такие процессы:

- получение,
- очистка,
- нормализация,
- превращение в оптимизированный набор данных.

Обычно это табличная форма, которая идеально подходит для этих методов, что были запланированы на этапе проектировки.

Многие проблемы могут возникнуть при появлении недействительных, двусмысленных или недостающих значений, повторении полей или данных, несоответствующих допустимому интервалу.

Изучение данных/визуализация

Изучение данных — это их анализ в графической или статистической репрезентации с целью поиска моделей или взаимосвязей. **Визуализация — лучший инструмент для выделения подобных моделей.**

За последние годы визуализация данных развилась так сильно, что стала независимой дисциплиной. Многочисленные технологии используются исключительно для отображения данных, а многие типы отображения работают так, чтобы получать только лучшую информацию из набора данных.

Исследование данных состоит из предварительного изучения, которое необходимо для понимания типа и значения собранной информации. Вместе с информацией, собранной при определении проблемы, такая категоризация определяет, какой метод анализа данных лучше всего подойдет для определения модели.

Эта фаза, в дополнение к изучению графиков, состоит из следующих шагов:

- Обобщение данных;
- Группировка данных;
- Исследование отношений между разными атрибутами;
- Определение моделей и тенденций;
- Построение моделей регрессионного анализа (классификации, кластеризации или других, подходящих для решения задачи);
- Интерпретация решения.

Как правило, анализ данных требует обобщения заявлений касательно изучаемых данных.

Обобщение — процесс, при котором количество данных для интерпретации уменьшается без потери важной информации.

Также используется группировка - кластерный анализ — метод анализа данных, используемый для поиска групп, объединенных общими атрибутами.

Еще один **важный этап анализа — идентификация отношений, тенденций и аномалий в данных**. Для поиска такой информации часто нужно использовать инструменты и проводить дополнительные этапы анализа, но уже на визуализациях.

Другие методы поиска данных, такие как деревья решений и ассоциативные правила, автоматически извлекают важные факты или правила из данных. Эти подходы используются параллельно с визуализацией для поиска взаимоотношений данных.

Предсказательная (предиктивная) модель

Предсказательная аналитика — это процесс в анализе данных, который нужен для создания или поиска подходящей статистической модели для предсказания вероятности результата.

После изучения данных у вас есть вся необходимая информация для развития математической модели, которая кодирует отношения между данными. Эти модели полезны для понимания изучаемой системы и используются в двух направлениях.

Первое — предсказания о значениях данных, которые создает система. В этом случае речь идет о регрессионных моделях.

Второе — классификация новых продуктов. Это уже модели классификации или модели кластерного анализа. На самом деле, можно разделить модели в соответствии с типом результатов, к которым те приводят:

- **Модели классификации**: если полученный результат — качественная переменная.
- **Регрессионные модели**: если полученный результат числовой.
- **Кластерные модели**: если полученный результат описательный.

Простые методы генерации этих моделей включают такие техники:

- линейная регрессия,
- логистическая регрессия,
- классификация,
- дерево решений,
- метод k-ближайших соседей.

Но таких методов много, и у каждого есть свои характеристики, которые делают их подходящими для определенных типов данных и анализа. Каждый из них приводит к появлению определенной модели, а их выбор соответствует природе модели продукта.

Некоторые из методов будут предоставлять значения, относящиеся к реальной системе и их структурам. Они смогут объяснить некоторые характеристики изучаемой системы простым способом. Другие будут делать хорошие предсказания, но их структура будет оставаться «черным ящиком» с ограниченной способностью объяснить характеристики системы.

Проверка модели

Проверка (валидация) модели, то есть фаза тестирования, — это важный этап. Он позволяет проверить модель, построенную на основе начальных данных. Он важен, потому что позволяет узнать достоверность данных, созданных моделью, сравнив их с реальной

системой. Но в этот раз вы берете за основу начальные данные, которые использовались для анализа.

Как правило, при использовании данных для построения модели вы будете воспринимать их как тренировочный набор данных (датасет), а для проверки — как валидационный набор данных.

Таким образом сравнивая данные, созданные моделью и созданные системой, вы сможете оценивать ошибки. С помощью разных наборов данных оценивать пределы достоверности созданной модели. Правильно предсказанные значения могут быть достоверны только в определенном диапазоне или иметь разные уровни соответствия в зависимости от диапазона учитываемых значений.

Этот процесс позволяет не только в числовом виде оценивать эффективность модели, но также сравнивать ее с другими. Есть несколько подобных техник; самая известная — перекрестная проверка (кросс-валидация). Она основана на разделении учебного набора на разные части. Каждая из них, в свою очередь, будет использоваться в качестве валидационного набора. Все остальные — как тренировочного. Так вы получите модель, которая постепенно совершенствуется.

Развертывание (деплой)

Это финальный шаг процесса анализа, задача которого — предоставить результаты, то есть выводы анализа. В процессе развертывания бизнес-среды анализ является выгодой, которую получит клиент, заказавший анализ. В технической или научной средах результат выдает конструкционные решения или научные публикации.

Развертывание — это процесс использования на практике результатов анализа данных.

Есть несколько способов развертывания результатов анализа данных или майнинга данных. **Обычно развертывание состоит из написания отчета для руководства или клиента.** Этот документ концептуально описывает полученные результаты. Он должен быть направлен руководству, которое будет принимать решения. Затем оно использует выводы на практике.

В документации от аналитика должны быть подробно рассмотрены следующие темы:

- Результаты анализа;
- Развертывание решения;
- Анализ рисков;
- Измерения влияния на бизнес.

Когда результаты проекта включают генерацию предсказательных моделей, они могут быть использованы в качестве отдельных приложений или встроены в ПО.

Количественный и качественный анализ данных

Данные можно поделить на две категории:

- Качественные (номинальные и порядковые);
- Количественные (дискретные и непрерывные);

Анализ данных полностью сосредоточен на данных. В зависимости от их происхождения можно проводить различия.

Когда анализируемые данные строго числовые или имеют структуру категорий, тогда речь идет о **количественном анализе**. Но если значения выражены описательно, естественным языком, то это **качественный анализ**.

Именно из-за разной природы данных, для анализа которых используются эти два типа анализа, можно и наблюдать их различия.

Количественный анализ

Количественные данные — значения наблюдений из измерений. Они могут быть **дискретными** и **непрерывными**. Дискретные значения можно посчитать, они отличны друг от друга. Непрерывные происходят от измерений или наблюдений, которые предполагают значения в заданном диапазоне.

Количественный анализ связан с данными, в которых наблюдается логический порядок. Данными, которые можно разбить на определенные категории. Это приводит к появлению структуры.

Порядок, классификация и структуры, в свою очередь, дают новую информацию и позволяют дальше обрабатывать ее более математическим путем. Это приводит к появлению моделей, которые создают количественные предсказания. А они уже позволяют специалисту в области анализа данных делать более объективные выводы.

Качественный анализ

Качественные данные — это значения наблюдений, которые можно разделить на группы или категории. Есть два типа качественных данных: **номинальные** и **порядковые**. Номинальная переменная не имеет внутреннего порядка, определенного в категории. Порядковая переменная, наоборот, имеет определенный порядок.

Качественный анализ работает с данными, в которых нет структуры или она не очевидна, а их природа — не числа и не какие-либо категории. Например, такие данные могут быть текстовыми, визуальными или звуковыми.

Такой анализ должен быть построен на методологиях, часто отличающихся от случая к случаю. Это позволяет извлекать информацию, которая создает модели, предлагающие качественные предсказания. А выводы на их основе будут включать субъективные интерпретации специалиста.

С другой стороны, качественный анализ может привести к исследованию новых систем и выводам, которые невозможны в случае со строгим математическим подходом. Часто такой вид анализа задействует изучение такие систем, как социальные феномены или сложных структур, не поддающихся измерению.

Машинное обучение: методы и способы

Подготовка данных для задач машинного обучения

Предварительная обработка и очистка данных должны проводиться до того, как набор данных будет использоваться для обучения модели. Необработанные данные зачастую искажены и ненадежны, и в них могут быть пропущены значения. Использование таких данных при моделировании может приводить к неверным результатам. Эти задачи являются частью процесса обработки и анализа данных группы и обычно подразумевают первоначальное изучение набора данных, используемого для определения и планирования необходимой предварительной обработки.

Назначение предварительной обработки и очистки данных

Реальные данные собираются для последующей обработки из разных источников и процессов. Они могут содержать ошибки и повреждения, негативно влияющие на качество набора данных. Вот какими могут быть типичные проблемы с качеством данных:

- **Неполнота**: данные не содержат атрибутов, или в них пропущены значения.
- **Шум**: данные содержат ошибочные записи или выбросы.
- **Несогласованность**: данные содержат конфликтующие между собой записи или расхождения.

Качественные данные - это необходимое условие для создания качественных моделей прогнозирования. Чтобы избежать появления ситуации «мусор на входе, мусор на выходе» и повысить качество данных и, как следствие, эффективность модели, необходимо провести мониторинг работоспособности данных, как можно раньше обнаружить проблемы и решить, какие действия по предварительной обработке и очистке данных необходимы.

Стандартные методы мониторинга работоспособности данных

Для проверки качества данных необходимо провести оценку:

- Количество **записей**.
- количество **атрибутов** (или **компонентов**);
- **Типы данных** атрибута (номинальные, порядковые или непрерывные).
- Количество **пропущенных значений**.
- Данные **правильного формата**.
    - Если данные имеют формат TSV или CSV, проверьте правильность разделения столбцов и строк соответствующими разделителями.
    - Если данные имеют формат HTML или XML, убедитесь, что формат данных соответствует надлежащим стандартам.
    - Для извлечения структурированной информации из частично структурированных или неструктурированных данных также может потребоваться синтаксический анализ.
- **Несогласованные записи данных**. Проверьте допустимость диапазона значений. Например, если данные содержат средний балл, проверьте, находится ли этот средний балл в обозначенном диапазоне (например, 0~4).

При обнаружении проблем с данными необходимо выполнить **обработку**, которая зачастую включает очистку пропущенных значений, нормализацию данных, дискретизацию, обработку текста для удаления и/или замены внедренных символов, которые могут влиять на выравнивание данных, смешанные типы данных в общих полях и пр.

Задачи предварительной обработки данных

- **Очистка данных** — восполнение пропущенных значений, обнаружение и удаление искаженных данных и выбросов.

- **Преобразование данных** — нормализация данных для снижения измерений и искажений.
- **Уплотнение данных** — создание выборки данных или атрибутов для упрощения обработки данных.
- **Дискретизация данных** — преобразование непрерывных атрибутов в категориальные, чтобы проще было использовать некоторые методы машинного обучения.
- **Очистка текста** — удаление внедренных символов, которые могут нарушать выравнивание данных, например внедренных символов табуляции в файле с разделителем-табуляцией, внедренных новых линий, которые могут, например, разбивать записи.

Обработка пропущенных значений

При работе с пропущенными значениями лучше сначала определить причину их появления в данных, что поможет решить проблему. Вот какие бывает методы обработки пропущенных значений:

- **Удаление**: удаление записей с пропущенными значениями.
- **Фиктивная подстановка** — замена пропущенных значений фиктивными, например, подстановка значения ***unknown*** (неизвестно) вместо категориальных или значения 0 вместо чисел.
- **Подстановка среднего значения**: пропущенные числовые данные можно заменить средним значением.
- **Подстановка часто используемого элемента**: пропущенные категориальные значения можно заменить наиболее часто используемым элементом.
- **Подстановка по регрессии**: использование регрессионного метода для замены пропущенных значений регрессионными.

Пропуски в категориальных признаках

Заполнить пропуски в категориальных признаках можно следующими способами:

- Заменить пропущенное значение новой категорией "Неизвестно".
- Заменить пропущенное значение наиболее популярным значением.

Пропуски в численных признаках

Если имеем дело с численными признаками, можно применить следующие подходы:

- Заменить пропущенное значение средним значением.
- Заменить пропущенное значение медианой. Если в данных присутствуют выбросы, этот способ замены пропусков является предпочтительным.

Выбросы

В данных могут присутствовать значения, являющиеся выбросами. Это, как правило, не ошибки. Однако, своими значениями они "шокируют" модель.

Для того, чтобы определить, является ли значение выбросом, пользуются характеристикой выборки, называемой интерквартильным размахом. Определяется он следующим образом:

$$IRQ = Q_3 - Q_1$$

где Q_1— первая квартиль — такое значение признака, меньше которого ровно 25% всех значений признаков. Q_3— третья квартиль — значение, меньше которого ровно 75% всех значений признака.

Для того, чтобы понять, является ли значение выбросом, можно воспользоваться эвристикой: выбросы лежат за пределами следующего интервала:

$$[Q_1 - 1.5 \cdot IRQ, Q_3 + 1.5 \cdot IRQ]$$

Чаще всего от выбросов в обучающей выборке лучше всего избавляться.

One-hot encoding

Это способ предварительной обработки категориальных признаков. Многие модели плохо работают с категориальными признаками как таковыми. Дело в том, что слово "Российская Федерация" нельзя просто взять и умножить на какое-нибудь число. Но многие модели работают именно так: берется коэффициент и на него умножается значение признака. Аналогичная операция выполняется с остальными признаками. Все результаты суммируются. На основе значения суммы делается вывод о принадлежности объекта к тому или иному классу (такие модели называются линейными).

Нормализация данных

Нормализация данных позволяет масштабировать числовые значения в указанном диапазоне. Ниже представлены распространенные методы нормализации данных.

- **Нормализация по методу минимакса**: линейное преобразование данных в диапазоне, например, от 0 до 1, где минимальное и максимальное масштабируемые значения соответствуют 0 и 1 соответственно.
- **Нормализация по Z-показателю**: масштабирование данных на основе среднего значения и стандартного отклонения: деление разницы между данными и средним значением на стандартное отклонение.
- **Десятичное масштабирование**: масштабирование данных путем удаления десятичного разделителя значения атрибута.

Нормализация — это приведение всех значений признака к новому диапазону. Например, к диапазону [0, 1]. Это полезно, поскольку значения признаков могут изменяться в очень большом диапазоне. Причем, значения разных признаков могут отличаться на несколько порядков. А после нормализации они все будут находиться в узком (и, часто, едином) диапазоне.

Наиболее популярным способом нормализации является нормализация методом минимакса. Для того, чтобы применить этот метод, должно быть известно максимальное и минимальное значение признака. Проблема в том, что эти значения известны не всегда.

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

Также довольно популярным методом является Z-нормализация. Диапазон новых значений для Z-нормализации выглядит следующим образом:

$$(-3\sigma[X], 3\sigma[X])$$

где $\sigma[X]$ — среднеквадратическое отклонение признака X.

Выполняется Z-нормализация по формуле ниже.

$$x_{new} = \frac{x_{old} - M[X]}{\sigma[X]}$$

где M[X] — математическое ожидание признака X.

Отметим, что в случае применения Z-нормализации к нескольким признакам, диапазон значений для них будет разным.

Дискретизация данных

Данные можно дискретизировать, преобразовав непрерывные значения в номинальные атрибуты или интервалы. Это можно сделать несколькими способами.

- **Группирование равной ширины**: разделение диапазона всех возможных значений атрибута в группы (N) одинакового размера с последующим присвоением значений, относящихся к ячейке с соответствующим номером.

- **Группирование равной высоты**: разделение всех возможных значений атрибута в группы (N), содержащие одинаковое количество экземпляров, с последующим присвоением значений, относящихся к ячейке с соответствующим номером.

Сокращение объема данных

Существуют различные методы, с помощью которых вы можете уменьшить размер данных для упрощения обработки данных. В зависимости от размера данных и домена вы можете применить такие методы:

- **Выборка записей**: создание выборки записей данных и выбор репрезентативного подмножества из общего набора данных.
- **Выборка атрибутов**: выбор в данных набора важнейших атрибутов.
- **Агрегирование**: разделение данных на группы и хранение числовых значений для каждой группы. Например, для уменьшения размера данных вы можете агрегировать числа, обозначающие ежедневный доход сети ресторанов за последние 20 лет, так, чтобы указывался ежемесячный доход.

Очистка данных

Текстовые поля в табличных данных могут содержать символы, сбивающие выравнивание столбцов или границы записей (или и то и другое вместе). Например, табуляции, внедренные в файл с разделителем-табуляцией, могут сбить выравнивание столбцов, а внедренные символы новой строки могут разорвать линии записей. Неправильная кодировка текста при его чтении или записи приводит к потере информации или появлению нечитаемых символов (например, нуль-символов), а также она может сказаться на разборе текста. Чтобы очистить текстовые поля, исправить выравнивание и извлечь структурированные текстовые данные из неструктурированных или полу-структурированных, могут потребоваться тщательные разбор и редактирование текста.

Функция просмотра данных позволяет ознакомиться с данными заблаговременно. Это поможет вам выявить те или иные проблемы с данными и применить соответствующие методы для решения этих проблем. Важно понимать, что породило проблемы, как они могли появиться. Это процесс поможет решить, к каким действиям по обработке данных нужно прибегнуть для устранения проблем. Определение окончательных вариантов использования и пользователей можно также использовать для установления приоритетов при обработке данных.

Способы машинного обучения

Способ	Общее описание	Области применения
Обучение с учителем (Supervised learning)	При обучении с учителем модель обучается на примерах. Аналитик обеспечивает алгоритм машинного обучения набором известных данных, который содержит необходимые входные и выходные значения. Алгоритм должен установить, как получаются по данным входам данные выходы. Сам аналитик знает решение поставленной задачи; алгоритм выявляет закономерности в данных, учится на основе наблюдений и делает прогнозы. Эти прогнозы затем корректируются оператором. Процесс продолжается до тех пор, пока алгоритм не достигнет высокого уровня точности/производительности.	**Классификация** В задачах классификации программа машинного обучения должна сделать заключение на основе наблюдённых значений и определить, к какой категории относятся новые наблюдения. – значения дискретны (принимают несколько заранее определённых значений, классов); – альтернативная постановка: предсказать вероятности определённых классов; – пример: предсказание пола (М/Ж), спам/не спам, вероятность ухода сотрудника/клиента; **Регрессия** В задачах регрессии программа машинного обучения должна оценить и понять взаимосвязи между переменными. Предмет регрессионного анализа – одна зависимая переменная и набор других изменяющихся переменных. Это делает анализ особенно полезным для прогнозирования и предсказаний. – значения у непрерывны (принимают любое значение из диапазона); – пример: прогнозирование уровня цены жилья по его описанию, предсказание объёма продаж; **Решающие деревья и случайный лес** Это k-ичное дерево с *решающими правилами* в нелистовых вершинах (узлах) и некотором заключении о целевой функции в листовых вершинах (*прогнозом*). В узлах находятся решающие правила и производится проверка соответствия примеров этому правилу по какому– либо атрибуту обучающего множества. Набор правил в дереве решений позволяет компактно описывать объекты; – задача: инструмент в системах поддержки принятия решений и интеллектуального анализа данных. **Ранжирование** – даны списки объектов и частичные порядки на этих списках (например, известно, какой элемент идёт за каким); – задача: построить ранжирующую модель, обобщающую способ ранжирования на новые данные;

Обучение без учителя (Unsupervised learning)	При обучении без учителя алгоритм исследует набор данных и выявляет скрытые закономерности корреляции между различными переменными. Этот способ можно использовать для группирования данных в кластеры на основании одних только их статистических свойств.	**Кластеризация** Кластеризация предполагает группирование наборов похожих данных (на основе определенных критериев). Это полезно для сегментации данных на несколько групп и проведении анализа на основе каждого набора данных по отдельности для поиска закономерностей. – объединить элементы в группы по их похожести (алгоритм кластеризации, используемый для вероятностного соединения записей); – пример: объединение данных из разнородных источников или по различным структурным подразделениям, чтобы построить общую картину клиентуры; домохозяйств по паттерну потребления электроэнергии; пользователей сервиса по поведению; кластеризация клиентов на группы. **Уменьшение размерности** Понижение размерности уменьшает количество используемых переменных и отделяет точную искомую информацию. – перевести данные в пространство меньшей размерности, с сохранением отношений между элементами; – пример: визуализация многомерных данных
Обучение с частичным привлечением учителя (Semi–Supervised learning)	Это гибрид обучения с учителем и без. Разметив небольшую часть данных, учитель дает машине понять, каким образом кластеризовать остальные. Обучение с частичным привлечением учителя включает обе проблемы: они используют маркированные, предопределённые и непредопределённые, немаркированные данные. Этот метод позволяет значительно повысить точность, поскольку мы можем использовать непредопределённые данные в тренировочном наборе данных с небольшим количеством маркированных предопределённых данных. Вариант обучения с учителем, в котором используются неразмеченные данные: – есть небольшой размеченный датасет (большой датасет дорог или его не всегда можно получить); – есть большой неразмеченный датасет	**Поиск аномалий** – поиск редких и необычных объектов, существенно отличающихся от основной массы; – пример: поиск мошеннических транзакций, т.е. распознавание мошенничеств с попытками выдать себя за другого. Мошенничества можно классифицировать как аномалию на фоне обычной активности для выявления попыток мошенничеств при онлайн-сделках.

	(неразмеченных данных много); – по неразмеченным данным модель обучает структуру и закономерности; – по размеченным данным на уже выделенной структуре учим модель с использованием разметки.	
Обучение с подкреплением (Reinforcement learning)	Акцент обучения с подкреплением делается на регламентированные процессы обучения, при которых алгоритм машинного обучения снабжен набором действий, параметров и конечных значений. Определив правила, алгоритм машинного обучения пытается изучить различные варианты и возможности, отслеживая и оценивая каждый раз результат, чтобы определить, какой из вариантов является оптимальным.	При обучении с подкреплением алгоритму позволяют взаимодействовать с окружением (например, сбрасывать бракованную продукцию с конвейера) и «вознаграждают», когда она правильно выполняет задание. Автоматизировав подсчет вознаграждений, можно дать возможность модели обучаться самостоятельно. – сортировка товаров в розничных магазинах. Некоторые продавцы экспериментируют с роботизированными системами сортировки предметов одежды, обуви и аксессуаров. Роботы, используя обучение с подкреплением и глубокое обучение, определяют, насколько сильно нужно сдавить предмет при хватании, и какой хват будет наилучшим. – разновидность этого способа, глубинное обучение с подкреплением, хорошо подходит для автономного принятия решений в случаях, когда возможностей обучения с учителем и без недостаточно.
Глубокое обучение (Deep learning)	Глубокое обучение может проходить как без учителя, так и с подкреплением. При глубоком обучении частично имитируются принципы обучения людей - используются нейронные сети для все более подробного уточнения характеристик набора данных.	Глубокие нейронные сети применяются при обработке множества изображений за короткое время и извлечения большего количества признаков, которые модель в конечном счете запоминает. Борьба с мошенничествами, так как улучшает точность распознавания благодаря автоматизации. Глубокое обучение может использоваться в автомобильной отрасли при выполнении ремонта и профилактического обслуживания.

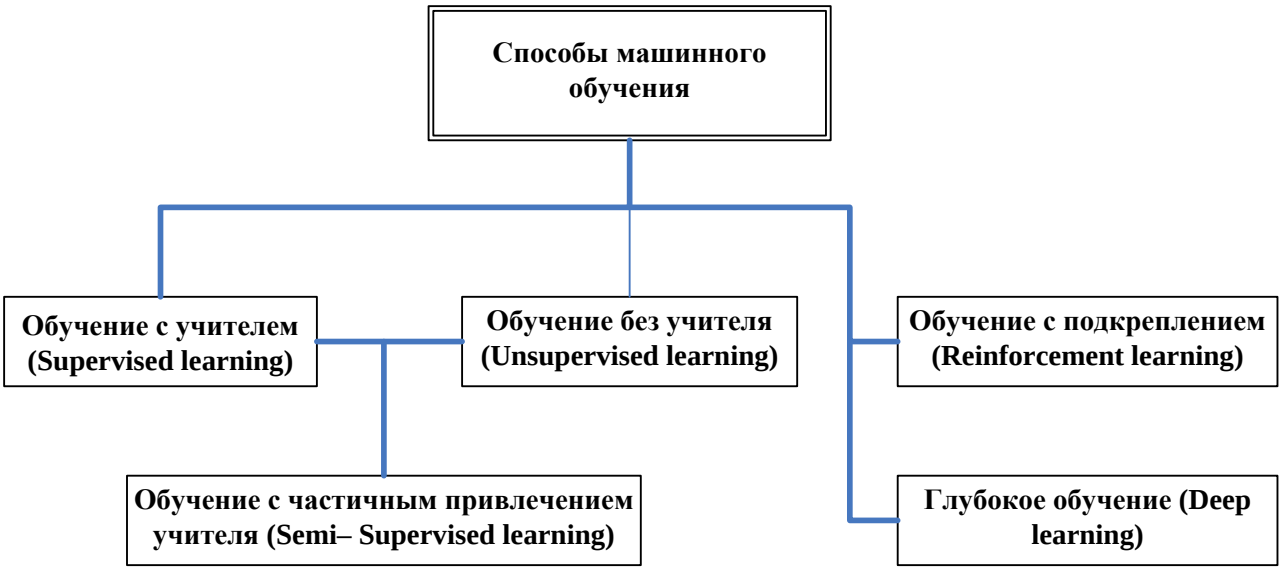


Рис.2 – Способы машинного обучения

Методы машинного обучения

Методы обучения с учителем

Метод	Общее описание	Области применения
Регрессия		
Линейная регрессия	Линейная регрессия – самая базовая разновидность регрессии. Простая линейная регрессия позволяет понять взаимосвязь между двумя непрерывными переменными.	
Нелинейная регрессия		
Классификация		
Классификация	Классификация - отнесение объектов предметной области к заранее определённым группам, называемым классами. При этом каждому классу должны принадлежать объекты, близкие по своим свойствам. Обобщая свойства известных объектов класса на новые, отнесённые к нему объекты, можно получать знания о них. Задача классификации решается с помощью аналитических моделей, называемых классификаторами. Классифицировать объект означает предъявить набор его признаков (обычно представленных в виде вектора) на вход модели–классификатора, которая должна присвоить ему метку или номер класса. В настоящее время разработано большое количество различных видов классификаторов, для построения которых используются как статистические методы (логистическая регрессия, дискриминантный анализ), так и методы машинного обучения (нейронные сети, деревья решений и др.).	Виды классификаторов: 1. Linear Classifier (LC) – Линейный классификатор. 2. Logistic Regression (LR) – Логистическая регрессия. 3. Support Vector Machines (SVM) – Метод опорных векторов. 4. Decision tree classifier (DT) – Дерево решений. 5. Random Forest Classifier (RF) – Случайный лес (используются деревья решений). 6. AdaBoost (Adaptive Boosting) Classifier (AB) – Адаптивное усиление. 7. Gaussian Naive Bayes (NB) – Гауссовский наивный байесовский классификатор. 8. Методы дискриминантного анализа (Discriminant Analysis).
Методы классификации		
Линейный классификатор	Линейный классификатор – алгоритм классификации, основанный на построении линейной разделяющей поверхности. Иными словами, это такой классификатор, дискриминантная функция которого линейна. В случае двух классов разделяющей поверхностью является гиперплоскость, которая делит пространство признаков на два	– анализ по набору данных спама, в котором задача состоит в том, чтобы классифицировать электронные письма как спам или не спам на основе набора функций, описывающих частоты слов, используемые в электронных письмах;

	полупространства. В случае большего числа классов разделяющая поверхность кусочно-линейна (как сумма выпуклых функций, которая тоже является выпуклой функцией).	
Преимущества и недостатки линейного классификатора **Достоинства**: – легко реализуется; – легко обобщается; – возможно динамическое (потоковое) обучение; – на сверхбольших выборках не обязательно брать все обучающие примеры; **Недостатки**: – возможна расходимость или медленная сходимость; – застревание в локальных минимумах; – проблема переобучения;		
Логистическая регрессия	Логистическая регрессия – метод, который определяет зависимости между переменными. Где одна из этих переменных категориально зависима, а остальные независимы. При этом применяется логистическая функция. На практике смысл логистической регрессии состоит в том, что она является достаточно мощным методом прогнозирования событий, которые из одной или несколько независимых переменных.	Это требуется в таких ситуациях как: замеры успешности проводимых рекламных кампаний; прогноз прибыли для конкретного товара; оценка возможности природного катаклизма
Преимущества и недостатки логистической регрессии **Преимущества:** – форма проста, а модель имеет хорошую интерпретируемость. Из веса функции мы видим важность функции и степень влияния на результат; – быстрое обучение; – результаты легко настраиваются. Выходным результатом логистической регрессии является вероятность возникновения, и результат классификации можно изменить, отрегулировав пороговое значение. **Недостатки:** – Эффект не подходит для несбалансированных выборок, и образцы должны быть обработаны. – Обработка нелинейных данных затруднительна. **Примечание.** – В моделировании лучше всего включать все важные независимые переменные, которые можно оценить с помощью метода пошагового скрининга. – Независимые переменные должны быть некоррелированными, то есть нет множественной коллинеарности. – Логистическая регрессия требует большего размера выборки, потому что когда размер выборки мал, эффект оценки максимального правдоподобия хуже, чем у метода наименьших квадратов.		
Метод опорных векторов (англ. support vector machine, SVM)	Данный метод изначально относится к бинарным классификаторам, хотя существуют способы заставить его работать и для задач мультиклассификации. Представляет из себя коллекцию алгоритмов, которые нужны для решения задач классификаций или регрессионного анализа. Опираясь на то, что объект,	Метод опорных векторов и его модифицированные версии помогают решать такие сложные задачи машинного обучения, как соединение ДНК, определение пола человека по фотографии, вывод рекламных баннеров на сайты.

	находящийся в многомерном пространстве, относится к одному из двух классов, метод опорных векторов строит гиперплоскость с мерностью на одну меньше, чем имеется в пространстве, чтобы все объекты оказались в одной из двух групп.	
Преимущества SVM: • Задача выпуклого квадратичного программирования хорошо изучена и имеет единственное решение. • Принцип оптимальной разделяющей гиперплоскости приводит к максимизации ширины разделяющей полосы, а, следовательно, к более уверенной классификации. **Недостатки классического SVM:** • Неустойчивость к шуму: выбросы в исходных данных становятся опорными объектами-нарушителями и напрямую влияют на построение разделяющей гиперплоскости. • Не описаны общие методы построения ядер и спрямляющих пространств, наиболее подходящих для конкретной задачи. • Нет отбора признаков.		
Дерево решений	Алгоритм дерева решений классифицирует объекты, отвечая на «вопросы» об их атрибутах, расположенные в узловых точках. В зависимости от ответа выбирается одна из ветвей, и так до тех пор, пока не будет достигнут «лист» - окончательный ответ.	Среди применений дерева решений - платформы управления знаниями для клиентского обслуживания, прогнозного назначения цен и планирования выпуска продукции. В страховой компании дерево решений поможет выяснить, какие виды страховых продуктов и премий лучше задействовать с учетом возможного риска. Используя данные о местонахождении и сведения о страховых случаях с учетом погодных условий, система может определять категории риска на основании поданных требований и затраченных сумм. Затем, используя модели, система будет оценивать новые заявления о страховой защите, классифицируя их по категории риска и возможному финансовому ущербу.
Случайный лес	Чтобы одиночное дерево решений давало точные результаты, его нужно обучать, алгоритм же случайного леса (random forest) использует «комитет» случайным образом созданных решающих деревьев с разными наборами атрибутов и дает возможность им проголосовать, чтобы выбрать самый популярный класс.	Случайный лес - универсальный, быстро обучаемый механизм для обнаружения связей внутри набора данных. В пример можно привести нежелательные массовые рассылки, создающие проблемы не только пользователям, но и провайдерам Интернета, которым из– за спама приходится иметь дело с повышенной нагрузкой на серверы. Для борьбы с проблемой были разработаны автоматизированные методы фильтрации спама, которые с помощью ансамбля решающих деревьев быстро и эффективно определяют нежелательные письма.

		Среди других применений - диагностика заболеваний путем анализа медицинской карты пациента, распознавание банковских мошенничеств, прогнозирование числа звонков в колл-центрах и прогнозирование вероятности прибыли и убытка при покупке определенных акций.
Преимущества и недостатки случайного леса **Преимущества:** - имеет высокую точность предсказания, на большинстве задач будет лучше линейных алгоритмов; точность сравнима с точностью бустинга; - практически не чувствителен к выбросам в данных из-за случайного сэмлирования; - не чувствителен к масштабированию (и вообще к любым монотонным преобразованиям) значений признаков, связано с выбором случайных подпространств; - не требует тщательной настройки параметров, хорошо работает «из коробки». С помощью «тюнинга» параметров можно достичь прироста от 0.5 до 3% точности в зависимости от задачи и данных; - способен эффективно обрабатывать данные с большим числом признаков и классов; - одинаково хорошо обрабатывает как непрерывные, так и дискретные признаки; - редко переобучается, на практике добавление деревьев почти всегда только улучшает композицию, но на валидации, после достижения определенного количества деревьев, кривая обучения выходит на асимптоту; - для случайного леса существуют методы оценивания значимости отдельных признаков в модели; - хорошо работает с пропущенными данными; сохраняет хорошую точность, если большая часть данных пропущена; - предполагает возможность сбалансировать вес каждого класса на всей выборке, либо на подвыборке каждого дерева; - вычисляет близость между парами объектов, которые могут использоваться при кластеризации, обнаружении выбросов или (путем масштабирования) дают интересные представления данных; - возможности, описанные выше, могут быть расширены до неразмеченных данных, что приводит к возможности делать кластеризацию и визуализацию данных, обнаруживать выбросы; - высокая параллелизуемость и масштабируемость. **Недостатки**: - в отличие от одного дерева, результаты случайного леса сложнее интерпретировать; - нет формальных выводов (p– values), доступных для оценки важности переменных; - алгоритм работает хуже многих линейных методов, когда в выборке очень много разреженных признаков (тексты, Bag of words); - случайный лес не умеет экстраполировать, в отличие от той же линейной регрессии (но это можно считать и плюсом, так как не будет экстремальных значений в случае попадания выброса); - алгоритм склонен к переобучению на некоторых задачах, особенно на зашумленных данных; - для данных, включающих категориальные переменные с различным количеством уровней, случайные леса предвзяты в пользу признаков с большим количеством уровней: когда у признака много уровней, дерево будет сильнее подстраиваться именно под эти признаки, так как на них можно получить более высокое значение оптимизируемого функционала (типа прироста информации); - если данные содержат группы коррелированных признаков, имеющих схожую значимость для меток, то предпочтение отдается небольшим группам перед большими; - больший размер получающихся моделей.		
Адаптивное усиление (AdaBoost)	Является мета-алгоритмом, в процессе обучения строит композицию из базовых алгоритмов обучения для улучшения их эффективности. AdaBoost является алгоритмом	Пример: для обнаружения лиц на изображениях. Алгоритм использует каскад отклонения, состоящий из множества слоев классификаторов. Когда окно обнаружения не

	адаптивного бустинга в том смысле, что каждый следующий классификатор строится по объектам, которые плохо классифицируются предыдущими классификаторами. AdaBoost вызывает слабый классификатор в цикле. После каждого вызова обновляется распределение весов, которые отвечают важности каждого из объектов обучающего множества для классификации. На каждой итерации веса каждого неверно классифицированного объекта возрастают, таким образом новый классификатор «фокусирует своё внимание» на этих объектах.	распознается на каком-либо слое как лицо, оно отклоняется. Первый классификатор в окне отбрасывает отрицательное окно, сводя вычислительные затраты к минимуму. Пример поступления студентов в университет, где они будут либо приняты, либо от них отказано. Здесь количественные и качественные данные можно найти с разных сторон. Например, результат приема, который может быть да / нет, может быть количественным, тогда как любая другая область, такая как навыки или хобби студентов, может быть качественной. Мы можем легко придумать правильную классификацию обучающих данных с большей вероятностью, чем шанс для условий, например, если ученик хорошо разбирается в конкретном предмете, то он / она допускается. Но трудно найти очень точный прогноз, и тогда слабые ученики входят в картину.
GBDT (Gradient Boost Decision Tree)	Другой метод усиления GBDT (градиентное повышение дерева решения), отличается от Adaboost, Каждый расчет GBDT состоит в том, чтобы уменьшить последний остаток, и есть новая модель в направлении остаточного снижения (отрицательный градиент).	
XGBoost (EXtreme Gradient Boosting)	Градиентный бустинг - это техника машинного обучения для задач классификации и регрессии, которая строит модель предсказания в форме ансамбля слабых предсказывающих моделей, обычно деревьев решений. Обучение ансамбля проводится последовательно в отличие, например, от бэггинга. На каждой итерации вычисляются отклонения предсказаний уже обученного ансамбля на обучающей выборке. Следующая модель, которая будет добавлена в ансамбль будет предсказывать эти отклонения. Таким образом, добавив предсказания нового дерева к предсказаниям обученного ансамбля мы можем уменьшить среднее отклонение модели, которое является целью оптимизационной задачи. Новые деревья добавляются в ансамбль до тех пор, пока ошибка	

	уменьшается, либо пока не выполняется одно из правил "ранней остановки".	
Преимущества и недостатки алгоритмов бустинга **Достоинства:** - хорошая обобщающая способность. В реальных задачах (не всегда, но часто) удаётся строить композиции, превосходящие по качеству базовые алгоритмы. Обобщающая способность может улучшаться (в некоторых задачах) по мере увеличения числа базовых алгоритмов. - простота реализации. - собственные накладные расходы бустинга невелики. Время построения композиции практически полностью определяется временем обучения базовых алгоритмов. - возможность идентифицировать объекты, являющиеся шумовыми выбросами. **Недостатки**: - AdaBoost склонен к переобучению при наличии значительного уровня шума в данных. Экспоненциальная функция потерь слишком сильно увеличивает веса наиболее трудных объектов, на которых ошибаются многие базовые алгоритмы. Однако именно эти объекты чаще всего оказываются шумовыми выбросами. В результате AdaBoost начинает настраиваться на шум, что ведёт к переобучению. Проблема решается путём удаления выбросов или применения менее агрессивных функций потерь. - AdaBoost требует достаточно длинных обучающих выборок. Другие методы линейной коррекции, в частности, бэггинг, способны строить алгоритмы сопоставимого качества по меньшим выборкам данных. - жадная стратегия последовательного добавления приводит к построению неоптимального набора базовых алгоритмов. Для улучшения композиции можно периодически возвращаться к ранее построенным алгоритмам и обучать их заново. Для улучшения коэффициентов можно оптимизировать их ещё раз по окончании процесса бустинга с помощью какого-нибудь стандартного метода построения линейной разделяющей поверхности. Рекомендуется использовать для этой цели SVM (машины опорных векторов). - бустинг может приводить к построению громоздких композиций, состоящих из сотен алгоритмов. Такие композиции исключают возможность содержательной интерпретации, требуют больших объёмов памяти для хранения базовых алгоритмов и существенных затрат времени на вычисление классификаций.		
Наивная байесовская классификация (НБК).	Данная классификация принадлежит семейству простых вероятностных классификаторов и возникает из теоремы Байеса. Другими словами, НБК предполагает, что наличие какого-нибудь признака в классе не связано с наличием какого-либо другого признака.	**Прогнозирование в реальном времени.** Благодаря простоте реализации и быстрым вычислениям его можно использовать для прогнозирования в режиме реального времени. **Предсказание нескольких классов** - Наивный байесовский алгоритм классификации можно использовать для прогнозирования апостериорной вероятности нескольких классов целевой переменной. **Классификация текста.** Благодаря многоуровневому прогнозированию, наивные байесовские алгоритмы классификации хорошо подходят для классификации текста. Вот почему он также используется для решения таких проблем, как фильтрация спама и анализ настроений. **Система рекомендаций** - наряду с такими алгоритмами, как совместная фильтрация, НБК создает систему

		рекомендаций, которую можно использовать для фильтрации невидимой информации и прогнозирования погоды, которой пользователь захочет получить данный ресурс или нет.
Преимущества и недостатки алгоритмов НБК **Достоинства** - наивная байесовская классификация легко реализуема и быстра. - сходиться быстрее, чем дискриминационные модели, такие как логистическая регрессия. - требует меньше тренировочных данных. - может делать вероятностные прогнозы и может обрабатывать как непрерывные, так и дискретные данные. - наивный байесовский алгоритм классификации можно использовать как для бинарных, так и для мультиклассовых задач классификации. **Недостатки**: - относительно низкое качество классификации в большинстве реальных задач. - независимость признаков, поскольку в реальной жизни практически невозможно иметь набор функций, которые полностью независимы друг от друга. - еще одна проблема, связанная с наивной байесовской классификацией, заключается в ее «нулевой частоте», которая означает, что если у категориальной переменной есть категория, но она не наблюдается в наборе обучающих данных, то наивная байесовская модель присвоит ей нулевую вероятность, и она не сможет прогнозирование.		
Методы дискриминантного анализа	Набор методов статистического анализа для решения задач распознавания образов, который используется для принятия решения о том, какие переменные разделяют (то есть «дискриминируют») возникающие наборы данных (так называемые «группы»). В дискриминантном анализе группы известны априори. Представление дискриминантного анализа состоит из статистических свойств данных, рассчитанных для каждого класса. Для каждой входной переменной это включает среднее значение для каждого класса и дисперсию, рассчитанную по всем классам. Предсказания производятся путём вычисления дискриминантного значения для каждого класса и выбора класса с наибольшим значением. Предполагается, что данные имеют нормальное распределение, поэтому перед началом работы рекомендуется удалить из данных аномальные значения. Выделяются: - линейный дискриминантный анализ (ЛДА); - квадратичный дискриминантный анализ (КДА); - дискриминантный анализ Фишера.	Виды задач дискриминантного анализа: - определение дискриминирующих признаков (т.е. признаков, которые позволяют отнести наблюдение к той или иной группе); - построение дискриминирующей функции; - прогнозирование будущих событий, связанных с попаданием объекта в ту или иную группу на основе значений его признака. Примеры: - образование: какие переменные относят выпускника средней школы к одной из трех категорий: поступающий в колледж, поступающий в высшую школу, отказывающийся от дальнейшего образования. - медицина: регистрация переменных, относящихся к состоянию больного, чтобы выяснить, какие переменные лучше показывают, что пациент, вероятно, выздоровел полностью, частично или совсем не выздоровел.

Методы обучения с учителем

Метод	Общее описание	Области применения
Кластеризация		
Кластеризация	Кластеризация - это группирование элементов данных, имеющих сходные характеристики, с помощью статистических алгоритмов. Это метод обучения без учителя, который можно использовать для решения задач классификации.	Примеры: сегментирование покупательской аудитории в зависимости от характеристик для уточнения адресации маркетинговых кампаний; рекомендации новостей конкретным читателям; помощь в работе правоохранительным органам. Кластеризация также действенна, когда в сложных наборах данных нужно обнаружить группы, которые трудно заметить без специальных средств. Примеры - от группирования похожих документов в базе данных до обнаружения по криминальным новостям территорий с повышенным уровнем преступности.
Методы кластеризации		
Метод k-ближайших соседей - k-nearest neighbors algorithm (KNN)	Метод относится к классу непараметрических, т.е. не требует предположений о том, из какого статистического распределения была сформирована обучающее множество. Алгоритм k-ближайших соседей оценивает, насколько велики шансы, что точка данных относится к той или иной группе. По существу, он просматривает точки данных вокруг одной выбранной точки и решает, к какой группе эта точка в реальности относится. Например, если точка расположена на сетке, а алгоритм пытается определить, к какой группе она относится (например, группа A или группа B), то он просмотрит точки рядом с данной и определит, к какой группе относится большинство этих точек.	

Преимущества и недостатки алгоритмов KNN

Достоинства:

- простота реализации.
- классификацию, проведенную данным алгоритмом, легко интерпретировать путём предъявления пользователю нескольких ближайших объектов.
- простота использования полученных результатов.
- решения не уникальны для конкретной ситуации, возможно их использование для других случаев.
- целью поиска является не гарантированно верное решение, а лучшее из возможных.

Недостатки:

- неэффективный расходу памяти и чрезмерное усложнение решающего правила вследствии необходимости хранения обучающей выборки целиком.
- поиск ближайшего соседа предполагает сравнение классифицируемого объекта со всеми объектами выборки, что требует линейного по длине выборки числа операций.

Метод k-средних – (k-means)	Наиболее простой, но в то же время достаточно неточный метод кластеризации в классической реализации. Он разбивает множество элементов векторного пространства на заранее известное число кластеров k. Действие алгоритма таково, что он стремится минимизировать среднеквадратичное отклонение на точках каждого кластера. Основная идея заключается в том, что на каждой итерации перевычисляется центр масс для каждого кластера, полученного на предыдущем шаге, затем векторы разбиваются на кластеры вновь в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике. Алгоритм завершается, когда на какой-то итерации не происходит изменения кластеров.	Применение алгоритма кластеризации K-средних • поведенческая сегментация; • обнаружение аномалий; • анализ социальной сети; • сегментация рынка.
Преимущества K-Means • Простота реализации • Масштабируемость до огромных наборов данных • Метод очень быстро обучается на новых примерах • Поддержка сложных форм и размеров. **Недостатки K-Means** • Чувствительность к выбросам • Трудоемкость выбора k • Уменьшение масштабируемости.		
Снижение размерности		
Снижение размерности	Под **уменьшением размерности** (англ. *dimensionality reduction*) в машинном обучении подразумевается уменьшение числа признаков набора данных. Наличие в нем признаков избыточных, неинформативных или слабо информативных может понизить эффективность модели, а после такого преобразования она упрощается, и соответственно уменьшается размер набора данных в памяти и ускоряется работа алгоритмов ML на нем. Уменьшение размерности может быть осуществлено методами выбора признаков (англ. *feature selection*) или выделения признаков (англ. *feature extraction*).	
Оберточные методы	**Оберточные методы** (англ. *wrapper methods*) находят подмножество искомых признаков последовательно, используя некоторый классификатор как источник оценки качества	

	выбранных признаков, т.е. этот процесс является циклическим и продолжается до тех пор, пока не будут достигнуты заданные условия останова. Оберточные методы учитывают зависимости между признаками, что является преимуществом по сравнению с фильтрами, к тому же показывают большую точность, но вычисления занимают длительное время, и повышается риск переобучения.	
Встроенные методы	Группа **встроенных методов** (англ. *embedded methods*) очень похожа на оберточные методы, но для выбора признаков используется непосредственно структуру некоторого классификатора. В оберточных методах классификатор служит только для оценки работы на данном множестве признаков, тогда как встроенные методы используют какую-то информацию о признаках, которую классификаторы присваивают во время обучения.	

Преимущества уменьшения размерности

- Это помогает в сжатии данных, и, следовательно, уменьшает пространство для хранения.
- Это уменьшает время вычислений.
- Это также помогает удалить избыточные функции, если таковые имеются.

Недостатки уменьшения размерности

- Это может привести к некоторой потере данных.
- PCA имеет тенденцию находить линейные корреляции между переменными, что иногда нежелательно.
- PCA терпит неудачу в случаях, когда среднее значение и ковариация недостаточны для определения наборов данных.
- Мы можем не знать, сколько основных компонентов следует придерживаться на практике, применяются некоторые правила большого пальца.

Обучение с подкреплением

Метод	Общее описание	Области применения
Нейронные сети	Нейронные сети имитируют структуру головного мозга: каждый искусственный нейрон соединяется с несколькими другими нейронами. Нейросети имеют многослойную структуру: нейроны на одном слое передают данные нескольким нейронам на следующем и т. д. В конечном счете данные достигают выходного слоя, где сеть выдает предположение о том, как решить задачу, классифицировать объект и т. п.	Нейросети применяются в целом ряде отраслей. В здравоохранении их используют при анализе медицинских снимков с целью ускорения диагностических процедур и поиска лекарств. В телекоммуникационной отрасли и медиаиндустрии нейросети можно применять для машинного перевода, распознавания мошенничеств и предоставления услуг виртуальных ассистентов. В финансовой отрасли их используют для распознавания мошенничеств,

		управления портфелями и анализа риска. В розничной торговле - для избавления от очередей в кассу и для персонализации обслуживания покупателей.

Глубокое обучение

Метод	Общее описание	Области применения
Глубокие нейронные сети	Нейронные сети имитируют структуру головного мозга: каждый искусственный нейрон соединяется с несколькими другими нейронами. Нейросети имеют многослойную структуру: нейроны на одном слое передают данные нескольким нейронам на следующем и т. д. В конечном счете данные достигают выходного слоя, где сеть выдает предположение о том, как решить задачу, классифицировать объект и т. п.	Нейросети применяются в целом ряде отраслей. В здравоохранении их используют при анализе медицинских снимков с целью ускорения диагностических процедур и поиска лекарств. В телекоммуникационной отрасли и медиаиндустрии нейросети можно применять для машинного перевода, распознавания мошенничеств и предоставления услуг виртуальных ассистентов. В финансовой отрасли их используют для распознавания мошенничеств, управления портфелями и анализа риска. В розничной торговле - для избавления от очередей в кассу и для персонализации обслуживания покупателей.

Программа на Python

Программа на языке *Python* может состоять из одного или нескольких **модулей**. Каждый *модуль* представляет собой *текстовый файл.*

Программа на *Python*, с точки зрения интерпретатора, состоит из **логических строк**. Одна логическая строка, как правило, располагается в одной физической, но длинные логические строки можно явно (с помощью обратной косой черты) или неявно (внутри скобок) разбить на несколько физических:

```
print (a, " - очень длинная строка, которая не помещается в", \
   80, "знакоместах")
```

Символ решетка (#) отмечает комментарий до конца строки.

Для многострочных комментариев применяются тройные парные кавычки:

```
"""
Многострочный комментарий
"""
```

Модули в Python

Модуль оформляется в виде отдельного файла с исходным кодом. *Стандартные модули* находятся в каталоге, где их может найти соответствующий *интерпретатор* языка.

Подключение модуля к программе на *Python* осуществляется с помощью оператора `import`. Унегоестьдвеформы: `import`и`from-import`:

```
import math, numpy as np
math.sqrt(25)
# import a function
from math import sqrt
print(sqrt(25))# no longer have to reference the module
from math import cos, floor
print(cos(0.25), floor(0.1))
# show all functions in numpy module
content = print(dir(np))
```

Типыданных

В языке программирования Python используется динамическое определение типа данных.

Как уже говорилось, все данные в *Python* представлены объектами. Имена являются лишь ссылками на эти объекты и не несут нагрузки по декларации типа. Значения встроенных типов имеют специальную поддержку в синтаксисе языка: можно записать **литерал** строки, числа, списка, кортежа, словаря (и их разновидностей). Синтаксическую же поддержку операций над встроенными типами можно легко сделать доступной и для объектов определяемых пользователями классов.

Следует также отметить, что объекты могут быть **неизменчивыми** и **изменчивыми**. Например, строки в *Python* являются неизменчивыми, поэтому *операции* над строками создают новые строки.

Python это объектно-ориентированный язык программирования, т.е.

1. Все данные в нем представляются объектами.
2. Каждый объект имеет собственную часть памяти и может состоять из других объектов.
3. Каждый объект имеет тип (принадлежит классу)
4. Все объекты одного типа могут принимать одни и те же сообщения (и выполнять одни и те же действия).

Для каждого объекта (число, функция, список, и т.д.) задан набор свойств (данных, хранящихся в объекте) и методов (способов использования объекта). Простые объекты (числа) могут рассматриваться традиционным образом, хотя на самом деле, операции над ними «+, -, *, /» и преобразование типов включены в описание их класса.

Типы данных	Функция преобразования	Пример присваивания	Примечания
Целый (integer)	`int()`	`a = 134`	
Большой целый(long)	`long()`	`a = 1L`	
Вещественный (float)	`float()`	`a = 1. a = 3.1415`	
Комплексный(complex)	`complex()`	`a = 1+2j`	
Логический (boolean)	`bool()`	`a = True a = False`	
Строка (string)	`str()`	`a='12414dfgas' a="123dfasdf" a="""rqw 1211dsfasf daffa"""`	поддержка многострочных выражений (тройные кавычки)
Список (list)	`list()`	`a = [12,'sd',1,b]` (b-имя переменной)	индексация (считывание, запись), добавление, исключение, упорядоченность, сортировка, поиск и перебор
Кортеж (tuple)	`tuple()`	`a = (12,'sd',1,b)` (b-имя переменной)	индексация (считывание, запись), упорядоченность, поиск и перебор
Словарь (dictionary)	`dict()`	`a={1:123,'df':13,'qw':b}` (b-имя переменной)	индексация (считывание, запись), добавление, исключение
Множество (set)	`set()`	`a = set(1,4,'34',b)` (b-имя переменной)	добавление, исключение, поиск и перебор

Списки, кортежи, множества, словари

Списки

Одним из основных типов данных в языке программирования Python являются списки (Lists) – индексированные совокупности переменных разного типа. Новый пустой список может быть создан с использованием любого из следующих операторов:

```
a = []
a = list()
```

Для создания заполненного списка, его значения указываются в квадратных скобках:

```
a = [1, 2, 3, "keyboard", "mouse", [0,1] ]
```

Ниже обобщены основные методы последовательностей. Следует напомнить, что последовательности бывают неизменчивыми и изменчивыми. У последних методов чуть больше.

Синтаксис	Семантика
`len(s)`	Длинапоследовательности`s`
`x in s`	Проверка принадлежности элемента последовательности. В новых версиях Python можно проверять принадлежность подстроки строке. Возвращает`True`или`False`
`x not in s`	= `not x in s`
`s + s1`	Конкатенацияпоследовательностей
`s*n`или`n*s`	Последовательность из `n` раз повторенной `s`. Если `n< 0`, возвращается пустая последовательность.
`s[i]`	Возвращает`i`-й элемент`s`или`len(s)+i`-й, если`i< 0`
`s[i:j:d]`	Срез из последовательности `s` от `i` до `j` с шагом `d` будет рассматриваться ниже
`min(s)`	Наименьшийэлемент`s`
`max(s)`	Наибольшийэлемент`s`

Кортежи

В языке программирования Python кортеж (Tuple) представляет собой неизменяемый список. Для него доступны все операции, что и для списков, за исключением влияющих на значения списка, например, добавление, изменение, удаление элементов и т.д. Кортежи записываются в круглых скобках и могут быть объявлены как:

```
a1 = tuple()
```

```
a2 = 1, 2, 3, "abc"
a3 = (1, 2, 3, "abc")
```

Основные операции с кортежами:

Функция или операция	Описание	Пример использования
`len(t)`	Определяется количество элементов кортежа t	

`t1 + t2`	Объединение кортежей. Получается новый кортеж, в котором после элементов кортежа t1 находятся элементы кортежа t2.	`t1=(1,2,3)`
		`t2=('raz','dva')`
		`t3=t1+t2`
		`t3 → (1, 2, 3, 'raz', 'dva')`

`t * n (или n * t)`	n-кратное повторение кортежа t	`t2=('raz','dva')` `t2*3 → ('raz', 'dva', 'raz', 'dva', 'raz', 'dva')`

`t[i]`	Выбор из t элемента с номером i, нумерация начинается с 0 (первый элемент имеет номер 0) Если i<0, отсчёт идёт с конца.	`t3= (1, 2, 3, 'raz', 'dva')`
		`t3[2] → 3`
		`t3[2] → 'raz'`

`t[i:j:k]`	Срез — кортеж, содержащий элементы кортежа t с номерами от i до j с шагом k (элемент с номером i входит в итоговый кортеж, а элемент с номером j уже не входит). Если k не указан (использован вариант t[i:j]), то элементы идут подряд (равносильно t[i:j:1]).	`t3= (1, 2, 3, 'raz', 'dva')` `t3[1:4] → (2, 3, 'raz')`
`min(t)`	Определяется элемент с наименьшим значением в соответствии с алфавитным порядком.	`t3= (1, 2, 3, 'raz', 'dva')` `min(t3) → 1`
`max(t)`	Определяется элемент с наибольшим значением в соответствии с алфавитным порядком.	`t3= (1, 2, 3, 'raz', 'dva')` `max(t3) → 'raz'`

Множества

Множество (Set) – это контейнер уникальных значений. Значения множества указываются в фигурных скобках. Множество создается с использованием функции **`set()`** или с указанием набора значений в фигурных скобках. Функция **`set()`** позволяет создать пустое множество, а также множество из строки, списка или другой итерируемой структуры данных. Если в наборе значений присутствуют повторяющиеся, все повторы будут удалены. Пустое множество не может быть создано с указанием пустых фигурных скобок. Для множеств не предусмотрено обращение к отдельному элементу. Например, для проверки принадлежности элемента **`element`**множеству **`set1`** и проходу по множеству, можно использовать:

```
if element in set1:
<операторы>
```

и

```
for element in set1:
<операторы>
```

Словари

Словари в языке программирования Python представляют собой неупорядоченные коллекции объектов, доступ к которым осуществляется по ключу. Альтернативные названия – ассоциативные массивы, хэш-таблицы.

Существует три способа создания словаря:

Указание пар "ключ-значение" в фигурных скобках: **`d = {"a": 1, "b": 2}`**

С помощью функции **dict()**:

```
d = dict(a = 1, b = 2)
d = dict([("a",1), ("b",2)])
```

Строки

Строки в языке программирования Python представляют собой списки символов. Они могут быть объявлены тремя способами:

с использованием одинарных кавычек: **'строка'**

с использованием двойных кавычек: **"строка"**

с использованием трех кавычек одинарного или двойного типа: **'''строка'''** или **"""строка"""**

Доступ к символам строки осуществляется аналогично доступу к элементам списка, используя позицию символа в строке.

NumPy Nd-array: создание массива, генерация и типы данных

NumericPython (NumPy) — это несколько модулей для вычислений с многомерными массивами, необходимых для многих численных приложений. Массив - это набор однородных элементов, доступных по индексам.

Особо подчеркнем отличие массива от набора данных (списка или кортежа). Величины, входящие в массив, имеют одинаковый тип и их количество жестко задается при инициализации. Элементы массива не являются объектами, это переменные в обычном понимании этого слова.

Основной элемент библиотеки NumPy — объект `ndarray` (что значит N-размерный массив). Этот объект является многомерным однородным массивом с заранее заданным количеством элементов. Однородный — потому что практически все объекты в нем одного размера или типа. На самом деле, тип данных определен другим объектом NumPy, который называется `dtype` (тип-данных). Каждый `ndarray` ассоциирован только с одним типом `dtype`.

Количество размерностей и объектов массива определяются его размерностью (`shape`), кортежем N-положительных целых чисел. Они указывают размер каждой размерности. Размерности определяются как оси, а количество осей — как ранг.

Еще одна странность массивов NumPy в том, что их размер фиксирован, а это значит, что после создания объекта его уже нельзя поменять. Это поведение отличается от такового у списков Python, которые могут увеличиваться и уменьшаться в размерах.

Простейший способ определить новый объект `ndarray` — использовать функцию `array()`, передав в качестве аргумента Python-список элементов.

```
import numpy as np
a =np.array([1,2,3])
print(a[1])
```

```
Результат:
2
```

Только что созданный массив имеет одну ось, а его ранг равняется `1`, то есть его форма — `(3,1)`. Для получения этих значений из массива необходимо использовать следующие атрибуты: `ndim` — для осей, `size` — для длины массива, `shape` — для его формы.

```
print(a.ndim)
print(a.size)
print(a.shape)
```

```
Результат:
1
3
(3,)
```

Это был пример простейшего одномерного массива. Но функциональность массивов может быть расширена и до нескольких размерностей. Например, при определении двумерного массива 2×2:

```
b = np.array([[1.3, 2.4],[0.3, 4.1]])
print(b.ndim)
print(b.size)
print(b.shape)
```

```
Результат:
2
4
(2, 2)
```

Ранг этого массива — 2, поскольку у него 2 оси, длина каждой из которых также равняется 2.

Создание массива

Есть несколько вариантов создания массива. Самый распространенный — список из списков, выступающий аргументом функции `array()`.

```
c=np.array([[1,2,3],[4,5,6]])
```

Функция `array()` также может принимать кортежи и последовательности кортежей.

```
d=np.array(((1,2,3),(4,5,6)))
```

Она также может принимать последовательности кортежей и взаимосвязанных списков.

```
e =np.array([(1,2,3),[4,5,6],(7,8,9)])

print(c)
```

```
Результат:
[[1 2 3]
 [4 5 6]]
```

```
print(d)
print(e)
```

```
[[1 2 3]
 [4 5 6]]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Функции генерации массива с заданными параметрами

Библиотека NumPy предоставляет набор функций, которые генерируют `ndarray` с начальным содержимым. Они создаются с разным значениями в зависимости от функции. Это очень полезная особенность. С помощью всего одной строки кода можно сгенерировать большой объем данных.

Функция **`zeros`**`()`, например, создает полный массив нулей с размерностями, определенными аргументом `shape`.

Например, для создания двумерного массива 3×3, можно использовать:

```
np.zeros((3,3))
```

```
([[0.,0.,0.],
[0.,0.,0.],
[0.,0.,0.]])
```

А функция **`ones`**`()` создает массив, состоящий из единиц.

```
np.ones((3,3))
```

```
([[1.,1.,1.],
[1.,1.,1.],
[1.,1.,1.]])
```

По умолчанию две функции создают массивы с типом данных `float64`. Полезная фишка — `arrange()`. Она генерирует массивы NumPy с числовыми последовательностями, которые соответствуют конкретным требованиям в зависимости от переданных аргументов. Например, для генерации последовательности значений между 0 и 10, нужно передать всего один аргумент — значение, которое закончит последовательность.

```
np.arange(0,10)
([0,1,2,3,4,5,6,7,8,9])
```

Если в начале нужен не ноль, то необходимо обозначить уже два аргумента: первый и последний.

```
np.arange(4,10)
([4,5,6,7,8,9])
```

Также можно сгенерировать последовательность значений с точным интервалом между ними. Если определено и третье значение в `arrange()`, оно будет представлять собой промежуток между каждым элементом.

```
np.arange(0,12,3)
([0,3,6,9])
```

Оно может быть и числом с плавающей точкой.

```
np.arange(0,6,0.6)
([0.,0.6,1.2,1.8,2.4,3.,3.6,4.2,4.8,5.4])
```

Пока что в примерах были только одномерные массивы. Для генерации двумерных массивов все еще можно использовать функцию `arrange()`, но вместе с `reshape()`. Она делит линейный массив на части способом, который указан в аргументе `shape`.

```
np.arange(0,12).reshape(3,4)
([[0,1,2,3],
[4,5,6,7],
[8,9,10,11]])
```

Похожаяна`arrange()`функция — `linspace()`. Она также принимает в качестве первых двух аргументов первое и последнее значения последовательности, но третьим аргументом является не интервал, а количество элементов, на которое нужно разбить последовательность.

```
np.linspace(0,10,5)
([0.,2.5,5.,7.5,10.])
```

Еще один способ получения массива — заполнение его случайными значениями. Это можно сделать с помощью функции `random()` из модуля `numpy.random`. Эта функция генерирует массив с тем количеством элементов, которые указаны в качестве аргумента.

```
np.random.random(3)
([0.78610272,0.90630642,0.80007102])
```

Полученные числа будут отличаться с каждым запуском. Для создания многомерного массива, нужно передать его размер в виде аргумента.

```
np.random.random((3,3))
([[0.07878569,0.7176506,0.05662501],
[0.82919021,0.80349121,0.30254079],
[0.93347404,0.65868278,0.37379618]])
```

Операции для работы с массивами

В NumPy реализовано много операций для работы с массивами:

- создание, модификация массива (изменение формы, транспонирование, поэлементные операции),
- выбор элементов,
- операции с массивами (различные типы умножения), сравнение массивов
- решение задач линейной алгебры (системы линейных уравнений, собственые вектора. собственные значения)
- создание наборов случайных данных
- быстрое преобразование Фурье

Создание массивов

В NumPy можно выделить три вида массивов:

- произвольные многомерные массивы (array)
- матрицы (matrix) – двухмерные квадратные массивы, для которых дополнительно определены операции возведения в степень и перемножения. Для работы с матрицами можно вместо “numpy” подключать “numpy.matrix”, в котором реализованы те же самые операции, только массивы – результаты операций будут приводится к типу “matrix”.
- сетки (grid) – массивы, в которых записаны значения координат точек сети (обычно ортогональной). Сетки позволяют удобно вычислять значение функций многих переменных.

Создание массивов из имеющихся данных

Для создания массивов существует множество функций. Самая распространенная из них **`array()`**.

```
>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
```

Команда	**Описание**
array(object[, dtype, copy, order, subok, ndmin])	Создать массив
asarray(a[, dtype, order])	Преобразовать в массив
ascontiguousarray(a[, dtype])	Размещает в памяти непрерывный массив(порядок данных как в Си)
asmatrix(data[, dtype])	Представить данные как матрицу
copy(a)	Возвращает копию объекта
frombuffer(buffer[, dtype, count,	Использует буфер, как одномерный массив

Команда	Описание
offset])	
fromfile(file[, dtype, count, sep])	Создает массив из данных файла
fromfunction(function, shape, **kwargs)	Создает и заполняет массив значениями функции от индексов элемента
fromiter(iterable, dtype[, count])	Создает одномерный массив из итератора
fromstring(string[, dtype, count, sep])	Создает одномерный массив из строки
loadtxt(fname[, dtype, comments, delimiter, …])	Создает массив из данных текстового файла

- a - объект или массив
- object - любой объект с упорядоченными данными
- dtype - тип данных (если не указан определяется по данным объекта)
- copy - да/нет, создать копию данных
- order - {‘C’, ‘F’, ‘A’}– порядок размещения элементов в памяти (Си, Фортран, любой)
- ndmin - минимальное число измерений (добавляет пустые массивы по недостающим измерениям)
- buffer - объект буфера
- count - число данных для чтения
- offset - отступ от начала
- file - объект файла
- sep - шаг чтения файла
- string - строка
- function - функция. Вызывается function(i,j,k,**kwargs), где i,j,k – индексы ячейки массива
- shape - форма массива
- **kwargs - словарь параметров для функции
- fname – имя файла
- comments – символ коментария
- delimiter – разделитель данных

Создание сеток

Команда	Описание	Пример
arange([start,] stop [, step,][, dtype])	Похоже на “range()”	`>>> np.arange(3.0)` `array([ 0.,  1.,  2.])`
linspace(start, stop [, num, endpoint, retstep])	Равномерный набор точек	`>>> np.linspace(2.0, 3.0, num=5)` `array([ 2., 2.25, 2.5, 2.75, 3.])`
logspace(start, stop [, num, endpoint, base])	Логарифмический набор точек	`>>> np.logspace(2.0, 3.0, num=4, base=2.0)` `array([ 4., 5.03968, 6.34960, 8.])`

meshgrid(x, y)	два вектора, описывающих точки ортогональной сетки.	`>>> X, Y = np.meshgrid([1,2,3], [4,5,7])`
		`>>> X`
		`array([[1, 2, 3],`
		`[1, 2, 3],`

Команда	Описание	Пример
		[1, 2, 3]]) >>> Y array([[4, 4, 4], [5, 5, 5], [7, 7, 7]])
mgrid	полный набор данных, описывающий многомерную равномерную ортогональную сетку (X,Y) или (X,Y,Z). Аргументы по каждому измерению: (start : stop : step). Если step – мнимое (5j) – то задается количество интервалов разбиения	>>> np.mgrid[0:5:3j,0:5:3j] array([[[0., 0., 0.], [2.5, 2.5, 2.5], [5., 5., 5.]], [[0., 2.5, 5.], [0., 2.5, 5.], [0., 2.5, 5.]]])
ogrid	сокращенный набор данных, описывающий многомерную равномерную ортогональную сетку (X,Y) или (X,Y,Z). Аргументы по каждому измерению: (start : stop : step). Если step – мнимое (5j) – то задается количество интервалов разбиения	>>> ogrid[0:5,0:5] [array([[0], [1], [2], [3], [4]]), array([[0, 1, 2, 3, 4]])]

- start – начало
- stop – окончание (для «arrange» по умолчанию НЕ включается, для остальных функций - включается)
- step – шаг
- num – число точек в выходном наборе
- endpoint – да/нет, включать крайнюю точку в набор данных
- retstep – да/нет, добавить в данные величину интервала
- x, y – одномерные массивы разбиения для осей.
- base – основание логарифма

Создание массивов определенного вида

Команда	Описание	Пример
empty(shape[, dtype, order]), empty_like(a[, dtype, order, subok])	выделяет место без инициализации (случайные числа)	>>> np.empty([2, 2], dtype=int) array([[-1073741821, -1067949133], [496041986, 19249760]])
eye(N[, M, k, dtype])	двухмерный диагональный со сдвигом	>>> np.eye(3, k=1) array([[0., 1., 0.], [0., 0., 1.], [0., 0., 0.]])
identity(N[, dtype])	единичная матрица (квадратная)	>>> np.identity(3) array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
ones(shape[, dtype,	все единицы	

Команда	Описание	Пример
order]) ones_like(a[, out])		
zeros(shape[, dtype, order]) zeros_like(a[, dtype, order, subok])	все нули	

tri(N[, M, k, dtype])	нижняя треугольная (из единиц)	`>>> np.tri(3, 5, 2, dtype=int)`
		`array([[1, 1, 1, 0, 0],`
		`[1, 1, 1, 1, 0],`
		`[1, 1, 1, 1, 1]])`

tril(a[, k])	вырезание нижней треугольной	`>>>`
		`np.tril([[1,2,3],[4,5,6],[7,8,9],`
		`[10,11,12]], -1)`
		`array([[ 0, 0, 0],`
		`[ 4, 0, 0],`
		`[ 7, 8, 0],`
		`[10, 11, 12]])`

triu(a[, k])	вырезание верхней треугольной	`>>>`
		`np.triu([[1,2,3],[4,5,6],[7,8,9],`
		`[10,11,12]], -1)`
		`array([[ 1, 2, 3],`
		`[ 4, 5, 6],`
		`[ 0, 8, 9],`
		`[ 0, 0, 12]])`

diag(a[, k])	вырезает диагональ или создает двумерную диагональную матрицу	
diagflat(a[, k])	двумерная диагональная матрица со всеми элементами из a.	

vander(x[, N])	создает определительВан Дер Монда	`>>> x = np.array([1, 2, 3, 5])`
		`>>> N = 3`
		`>>> np.vander(x, N)`
		`array([[ 1, 1, 1],`
		`[ 4, 2, 1],`
		`[ 9, 3, 1],`
		`[25, 5, 1]])`

mat(data[, dtype])	преобразует данные в матрицу	
bmat(obj[, ldict, gdict])	создает матрицу из строки, последовательности или массива	

- shape – форма массива
- dtype – тип данных
- order – порядок размещения данных(Си, Фортран)
- a – объект типа массива
- N – число строк
- M – число столбцов

- k – задает диагональ (к=0 – главная, к>0 – смещение вверх, к<0 – смещение вниз)
- x – одномерный массив или список

Трансформации массива без изменения элементов

Команда	**Описание**
resize(a, new_shape)	возвращает новый массив заданной формы (если элементов не хватает, то заполняется циклически)
reshape(a, newshape[, order])	новая форма для данных (полный размер обязан совпадать)
ravel(a[, order])	возвращает новый одномерный массив
ndarray.flat	итератор по массиву (вызывается как метод)
ndarray.flatten([order])	копия массива без формы (вызывается как метод)
rollaxis(a, axis[, start])	сдвигает выбранную ось до нужного положения
swapaxes(a, axis1, axis2)	меняет две оси в массиве
ndarray.T	То же что и транспонирование. (если размерность=1, то не изменяется)
transpose(a[, axes])	транспонирует массив (переставляет измерения)
fliplr(a)	симметрично отображает массив относительно вертикальной оси (право-лево)
flipud(a)	симметрично отображает массив относительно горизонтальной оси (верх-низ)
roll(a, shift[, axis])	циклический сдвиг элементов вдоль выбранного направления
rot90(a[, k])	поворот массива против часовой стрелке на 90 градусов
tile(A, repeats)	создает матрицу повторением заданной определенное количество
repeat(a, repeats[, axis])	повторяет элементы массива

- a
- newshape
- order
- start
- axis1, axis2
- shift
- k
- repeats

Слияние и разделение массивов

Команда	**Описание**
column_stack(tup)	собирает одномерные массивы -столбцы в двухмерный
concatenate(tup[, axis])	соединяет последовательность массивов вместе
dstack(tup)	собирает массивы «по глубине» (по третьей оси).
hstack(tup)	собирает массивы «по горизонтали» (по столбцам).
vstack(tup)	собирает массивы «по вертикали» (по строкам).
array_split(a, indices_or_sections[, axis])	разделяет массив по порциям
dsplit(a, indices_or_sections)	разделяет массив «по глубине» (по третьей оси)
hsplit(a, indices_or_sections)	разделяет массив «по горизонтали» (по столбцам).
split(a, indices_or_sections[, axis])	разделяет массив на части равной длины
vsplit(a, indices_or_sections)	разделяет массив «по вертикали» (по строкам).

- tup – кортеж массивов
- axis – ось
- a – массив
- indices_or_sections – размер порции при разделении

Функции, определенные для массивов

Алгебраические функции

Функция	**Описание**
isreal(x)	проверка на действительность (по элементам)
iscomplex(x)	проверка на комплексность (по элементам)
isfinite(x[, out])	приверка элементов на числовое значение (не бесконечность и не «не число»).
isinf(x[, out])	проверка на бесконечность (по элементам)
isnan(x[, out])	проверка аргумента на «не число» (NaN), результат – логический массив
signbit(x[, out])	истина, если установлен бит знака (меньше нуля)
copysign(x1, x2[, out])	меняет знак x1 на знак x2 (по элементам
nextafter(x1, x2[, out])	следующее в направлении x2 число, представимое в виде с плавающей точкой (по элементам)
modf(x[, out1, out2])	дробная и целая часть числа
ldexp(x1, x2[, out])	вычисляет y = x1 * 2**x2.
frexp(x[, out1, out2])	разделение числа на нормированную часть и степень
absolute(x[, out])	Calculate the absolute value element-wise.
rint(x[, out])	округление элементов массива
trunc(x[, out])	отбрасывание дробной части (по элементам)
floor(x[, out])	целая часть
ceil(x[, out])	минимальное целое большее числа
sign(x[, out])	знаки элементов
conj(x[, out])	комплексное сопряжение (по элементам).
exp(x[, out])	экспонента (по элементам)
exp2(x[, out])	2**элемент (по элементам)
log(x[, out])	натуральный логарифм (по элементам)
log2(x[, out])	двоичный логарифм (по элементам)
log10(x[, out])	десятичный логарифм (по элементам)
expm1(x[, out])	exp(x) - 1 (по элементам)
log1p(x[, out])	Return the natural logarithm of one plus the input array, element-wise.
sqrt(x[, out])	квадратный корень (для положительных) (по элементам)
square(x[, out])	квадрат (по элементам)
reciprocal(x[, out])	обратная величина (по элементам)

- x – массив
- out – место для результата

Тригонометрические функции

Все тригонометрические функции работают с радианами.

Функция	Обратная функция	Описание
sin(x[, out])	arcsin(x[, out])	синус (по элементам)
cos(x[, out])	arccos(x[, out])	косинус (по элементам)
tan(x[, out])	arctan(x[, out])	тангенс (по элементам)
	arctan2(x1, x2[, out])	арктангенс x1/x2 с правильным выбором четверти (по элементам)
hypot(x1, x2[, out])		гипотенуза по двум катетам (по элементам)
sinh(x[, out])	arcsinh(x[, out])	гиперболический синус (по элементам)
cosh(x[, out])	arccosh(x[, out])	гиперболический косинус (по элементам)
tanh(x[, out])	arctanh(x[, out])	гиперболический тангенс (по элементам)
deg2rad(x[, out])	rad2deg(x[, out])	преобразование градусов в радианы (по элементам)

- x, x1, x2 – массивы
- out – место для результата

Функции двух аргументов (бинарные функции)

Для правильной работы с логическими бинарными функциям (AND, OR) необходимо явно их записывать через функции модуля «NumPy», а не полагаться на встроенные функции питона.

Функция	Описание
add(x1, x2[, out])	сумма (по элементам)
subtract(x1, x2[, out])	разность (по элементам)
multiply(x1, x2[, out])	произведение (по элементам)
divide(x1, x2[, out])	деление (по элементам)
logaddexp(x1, x2[, out])	логарифм суммы экспонент (по элементам)
logaddexp2(x1, x2[, out])	логарифм по основанию 2 от суммы экспонент (по элементам)
true_divide(x1, x2[, out])	истинное деление (с преобразованием типов)
floor_divide(x1, x2[, out])	деление без преобразования типов (целочисленное)
negative(x[, out])	обратные элементы (по элементам)
power(x1, x2[, out])	элементы первого массива в степени элементов из второго массива (по элементам)
remainder(x1, x2[, out]), mod(x1, x2[, out]), fmod(x1, x2[, out])	остаток от деления (по элементам).
greater(x1, x2[, out])	истина, если (x1 > x2) (по элементам).
greater_equal(x1, x2[, out])	истина, если (x1 > =x2) (по элементам).
less(x1, x2[, out])	истина, если (x1 < x2) (по элементам).
less_equal(x1, x2[, out])	истина, если (x1 =< x2) (по элементам).
not_equal(x1, x2[, out])	истина, если (x1 != x2) (по элементам).
equal(x1, x2[, out])	истина, если (x1 == x2) (по элементам).
logical_and(x1, x2[, out])	истина, если (x1 AND x2) (по элементам).
logical_or(x1, x2[, out])	истина, если (x1 OR x2) (по элементам).
logical_xor(x1, x2[, out])	истина, если (x1 XOR x2) (по элементам).
logical_not(x[, out])	истина, если (NOT x1) (по элементам).
maximum(x1, x2[, out])	максимум из элементов двух массивов(по элементам).

- x1, x2 – массивы
- out – место для результата

Бинарные функции поддерживают дополнительные методы, позволяющие накапливать значения результата различными способами.

- accumulate() - Аккумулирование результата.
- outer() - Внешнее «произведение».
- reduce() - Сокращение.
- reduceat() - Сокращение в заданных точках.

Методы accumulate(), reduce() и reduceat() принимают необязательный аргумент - номер размерности, используемой для соответствующего действия. По умолчанию применяется нулевая размерность.

Другие функций для массивов

Функция	**Описание**
apply_along_axis(func1d, axis, a, *args)	Применить функцию к одномерному срезу вдоль оси
apply_over_axes(func, a, axes)	применить функцию последовательно вдоль осей.
vectorize(pyfunc[, otypes, doc])	обобщить функцию на массивы
frompyfunc(func, nin, nout)	берет произвольную функцию Python и возвращает функцию Numpy
piecewise(a, condlist, funclist, *args, **kw)	применение кусочно-определенной функции к массиву

- func1d – функция для вектора
- func – скалярная функция
- axis – индекс оси
- arr – массив
- *args, **kw – дополнительные аргументы
- nin – число входных параметров
- nout – число выходных параметров
- condlist - список условий
- funclist – список функций (для каждого условия)

Сортировка, поиск, подсчет

Команда	**Описание**
sort(a[, axis, kind, order])	отсортированная копия массива
lexsort(keys[, axis])	Perform an indirect sort using a sequence of keys.
argsort(a[, axis, kind, order])	аргументы, которые упорядочивают массив
array.sort([axis, kind, order])	сортирует массив на месте (метод массива)
msort(a)	копия массива отсортированная по первой оси
sort_complex(a)	сортировка комплексного массива по действительной части, потом по мнимой
argmax(a[, axis])	индексы максимальных значений вдоль оси
nanargmax(a[, axis])	индексы максимальных значений вдоль оси (игнорируются NaN).
argmin(a[, axis])	индексы минимальных значений вдоль оси
nanargmin(a[, axis])	индексы минимальных значений вдоль оси (игнорируются NaN).
argwhere(a)	массив индексов ненулевых элементов. данные сгруппированы по

Команда	Описание
	элементам([x1,y1,..],[x2,y2,..]….)
nonzero(a)	массивы индексов ненулевых элементов. сгруппированы по размерностям (индексы X, индексы Y, т.д.)
flatnonzero(a)	индексы ненулевых элементов в плоской версии массива
where(condition, [x, y])	возвращает массив составленный из элементов x (если выполнено условие) и y (в противном случае). Если задано только condition, то выдает его «не нули».
searchsorted(a, v[, side])	индексы мест, в которые нужно вставить элементы вектора для сохранения упорядоченности массива
extract(condition, a)	возвращает элементы (одномерный массив), по маске (condition)
count_nonzero(a)	число ненулевых элементов в массиве

- a – массив
- axis – индекс оси для сортировки (по умолчанию «–1» - последняя ось)
- kind – {‘quicksort’, ‘mergesort’, ‘heapsort’} тип сортировки
- order – индексы элементов, определяющие порядок сортировки
- keys – (k,N) массив из k элементов размера (N). k “колонок” будут отсортированы. Последний элемент – первичный ключ для сортировки.
- condition – матрица условий (маска)
- x, y – массивы для выбора элементов
- v – вектор
- side – {‘left’, ‘right’} позиция для вставки элемента (слева или справа от найденного индекса)

Дискретное преобразование Фурье (numpy.fft)

Прямое преобразование	Обратное преобразование	Описание
fft(a[, s, axis])	ifft(a[, s, axis])	одномерное дискретное преобразование Фурье
fft2(a[, s, axes])	ifft2(a[, s, axes])	двумерное дискретное преобразование Фурье
fftn(a[, s, axes])	ifftn(a[, s, axes])	многомерное дискретное преобразование Фурье
rfft(a[, s, axis])	irfft(a[, s, axis])	одномерное дискретное преобразование Фурье (действительные числа)
rfft2(a[, s, axes])	irfft2(a[, s, axes])	двумерное дискретное преобразование Фурье (действительные числа)
rfftn(a[, s, axes])	irfftn(a[, s, axes])	многомерное дискретное преобразование Фурье (действительные числа)
hfft(a[, s, axis])	ihfft(a[, s, axis])	преобразование Фурье сигнала с Эрмитовым спектром
fftfreq(n[, d])		частоты дискретного преобразования Фурье
fftshift(a[, axes])	ifftshift(a[, axes])	преобразование Фурье со сдвигом нулевой компоненты в центр спектра

- a - массив
- s – число элементов вдоль каждого направления преобразования (если больше размерности, то дополняются нулями)
- axes – последовательность осей для преобразования
- n – ширина окна
- d – шаг по частоте при выводе

Линейная алгебра (numpy.linalg)

Модуль numpy.linalg содержит алгоритмы линейной алгебры, в частности нахождение определителя матрицы, решений системы линейных уравнений, обращение матрицы, нахождение собственных чисел и собственных векторов матрицы, разложение матрицы на множители: Холецкого, сингулярное, метод наименьших квадратов и т.д.

Команда	**Описание**
dot(a, b[, out])	скалярное произведение массивов
vdot(a, b)	векторное произведение векторов
inner(a, b)	внутреннее произведение массивов
outer(a, b)	внешнее произведение векторов
tensordot(a, b[, axes])	тензорное скалярное произведение вдоль оси (размерность больше 1)
einsum(subscripts, *operands[, out, dtype, …])	суммирование Эйншнейна Evaluates the Einstein summation convention on the operands.
linalg.matrix_power(M, n)	возведение квадратной матрицы в степень n
kron(a, b)	произведение Кронекера двух массивов
linalg.norm(a[, ord])	норма матрицы или вектора.
linalg.cond(a[, ord])	число обусловленности матрицы.
linalg.det(a)	определитель
linalg.slogdet(a)	знак и натуральный логарифм определителя
trace(a[, offset, axis1, axis2, dtype, out])	сумма элементов по диагонали.
linalg.cholesky(a)	разложение Холецкого
linalg.qr(a[, mode])	разложение QR
linalg.svd(a[, full_matrices, compute_uv])	сингулярное разложение
linalg.solve(a, b)	решение линейного матричного уравнения или системы скалярных уравнений.
linalg.tensorsolve(a, b[, axes])	решение тензорного уравнения a x = b для x.
linalg.lstsq(a, b[, rcond])	решение матричного уравнения методом наименьших квадратов
linalg.inv(a)	обратная матрица (для умножения)
linalg.pinv(a[, rcond])	псевдо-обратная матрица (Мура-Пенроуза)
linalg.tensorinv(a[, ind])	«обратный» к многомерному массиву
linalg.eig(a)	собственные значения и правые собственные вектора квадратной
linalg.eigh(a[, UPLO])	собственные значения и собственные вектора эрмитовой или симметричной матрицы
linalg.eigvals(a)	собственные значения произвольной матрицы
linalg.eigvalsh(a[, UPLO])	собственные значения эрмитовой или действительной симметричной матрицы

- a, b - матрицы
- out - место для результата
- ord – определяет способ вычисления нормы
- axes – массив осей для суммирования
- axis – индекс оси
- subscripts – индексы для суммирования

- *operands – список массивов
- dtype – тип результата
- offset – положение диагонали
- mode - {‘full’, ‘r’, ‘economic’} – выбор алгоритма разложения
- full_matrices – составлять полные матрицы
- compute_uv – выводить все матрицы
- rcond – граница для отбрасывания маленьких собственных значений
- ind – число индексов для вычисления обратной
- UPLO - {‘L’, ‘U’} выбирает часть матрицы для работы

Случайные величины (numpy.random)

В модуле numpy.random собраны функции для генерации массивов случайных чисел различных распределений и свойств. Их можно применять для математического моделирования. Функция random() создает массивы из псевдослучайных чисел, равномерно распределенных в интервале (0, 1). Функция RandomArray.randint() для получения массива равномерно распределенных чисел из заданного интервала и заданной формы. Можно получать и случайные перестановки с помощью RandomArray.permutation(). Доступны и другие распределения для получения массива нормально распределенных величин с заданным средним и стандартным отклонением:

Следующая таблица приводит основные функции модуля.

seed([seed])- перезапуск генератора случайных чисел

Команда	**Описание**
rand(d0, d1, …, dn)	набор случайных чисел заданной формы
randn([d1, …, dn])	набор (или наборы) случайных чисел со стандартным нормальным распределением
randint(low[, high, size])	случайные целые числа от low (включая) до high (не включая).
random_integers(low[, high, size])	случайные целые числа между low и high (включая).
random_sample([size])	случайные рациональные числа из интервала [0.0, 1.0).
bytes(length)	случайные байты
shuffle(x)	тасовка элементов последовательностина месте
permutation(x)	возвращает последовательность, переставленных случайным образом элементов
seed([seed])	перезапуск генератора случайных чисел
beta(a, b[, size])	числа с Бетта- распределением f(x,α,β)=1B(α,β)xα−1(1−x)β−1f(x,α,β)=1B(α,β)xα−1(1−x)β−1 B(α,β)=∫10tα−1(1−t)β−1dtB(α,β)=∫01tα−1(1−t)β−1dt
binomial(n, p[, size])	числа с биномиальным распределением P(N)=(nN)pN(1−p)n−NP(N)=(nN)pN(1−p)n−N
chisquare(df[, size])	числа с распределением хи-квадрат p(x)=(1/2)k/2Γ(k/2)xk/2−1e−x/2p(x)=(1/2)k/2Γ(k/2)xk/2−1e−x/2
mtrand.dirichlet(alpha[, size])	числа с распределением Дирихле (alpha – массив параметров).
exponential([scale, size])	числа с экспоненциальным распределением f(x,1β)=1βexp(−xβ)f(x,1β)=1βexp(−xβ)
f(dfnum, dfden[, size])	числа с F распределением (dfnum – число степеней свободы числителя > 0; dfden –число степеней свободы знаменателя >0.)
gamma(shape[, scale, size])	числа с Гамма - распределением
geometric(p[, size])	числа с геометрическим распределением

Команда	Описание
gumbel([loc, scale, size])	числа с распределением Гумбеля
hypergeometric(ngood, nbad, nsample[, size])	числа с гипергеометрическим распределением (n = ngood, m = nbad, and N = number of samples)
laplace([loc, scale, size])	числа с распределением Лапласа
logistic([loc, scale, size])	числа с логистическим распределением
lognormal([mean, sigma, size])	числа с логарифмическим нормальным распределением
logseries(p[, size])	числа с распределением логарифмического ряда
multinomial(n, pvals[, size])	числа с мультиномиальным распределением
multivariate_normal(mean, cov[, size])	числа с мульти нормальным распределением (mean – одномерный массив средних значений; cov – двухмерный симметричный, полож. определенный массив (N, N) ковариаций
negative_binomial(n, p[, size])	числа с отрицательным биномиальным распределением
noncentral_chisquare(df, nonc[, size])	числа с нецентральным распределением хи-квадрат
noncentral_f(dfnum, dfden, nonc[, size])	числа с нецентральным F распределением (dfnum - целое > 1; dfden – целое > 1; nonc : действительное >= 0)
normal([loc, scale, size])	числа с нормальным распределением
pareto(a[, size])	числа с распределением Паретто
poisson([lam, size])	числа с распределением Пуассона
power(a[, size])	числа со степенным распределением [0, 1]
rayleigh([scale, size])	числа с распределением Релея
standard_cauchy([size])	числа со стандартным распределением Коши
standard_exponential([size])	числа со стандартным экспоненциальным распределением
standard_gamma(shape[, size])	числа с гамма- распределением
standard_normal([size])	числа со стандартным нормальным распределением (среднее=0, сигма=1).
standard_t(df[, size])	числа со стандартным распределением Стьюдента с df степенями свободы
triangular(left, mode, right[, size])	числа из треугольного распределения
uniform([low, high, size])	числа с равномерным распределением
vonmises(mu, kappa[, size])	числа с распределением Майсеса (I- модифицированная функция Бесселя)
wald(mean, scale[, size])	числа с распределением Вальда
weibull(a[, size])	числа с распределением Вайбулла
zipf(a[, size])	числа с распределением Зипфа (зетта функция Римана)

- size - число элементов по каждому измерению

Статистика

Команда	Описание
amin(a[, axis, out])	минимум в массиве или минимумы вдоль одной из осей
amax(a[, axis, out])	максимум в массиве или максимумы вдоль одной из

Команда	Описание
	осей
nanmax(a[, axis])	максимум в массиве или максимумы вдоль одной из осей (игнорируются NaN).
nanmin(a[, axis])	минимум в массиве или минимумы вдоль одной из осей (игнорируются NaN).
ptp(a[, axis, out])	диапазон значений (максимум - минимум) вдоль оси
average(a[, axis, weights, returned])	взвешенное среднее вдоль оси
mean(a[, axis, dtype, out])	арифметическое среднее вдоль оси
median(a[, axis, out, overwrite_input])	вычисление медианы вдоль оси
std(a[, axis, dtype, out, ddof])	стандартное отклонение вдоль оси
corrcoef(x[, y, rowvar, bias, ddof])	коэффициенты корреляции
correlate(a, v[, mode, old_behavior])	кросс-корреляция двух одномерных последовательностей
cov(m[, y, rowvar, bias, ddof])	ковариационная матрица для данных
histogram(a[, bins, range, normed, weights, …])	гистограмма из набора данных
histogram2d(x, y[, bins, range, normed, weights])	двумерная гистограмма для двух наборов данных
histogramdd(sample[, bins, range, normed, …])	многомерная гистограмма для данных
bincount(x[, weights, minlength])	число появление значения в массиве неотрицательных значений
digitize(x, bins)	возвращает индексы интервалов к которым принадлежат элементы массива

- a – массив
- axis – индекс оси
- out – место для результата
- weights- веса
- returned – дополнительно выдать сумму весов
- dtype – тип данных для накопления суммы
- overwrite_input – использовать входящий массив для промежуточных вычислений
- ddof- дельта степеней свободы (см. описание)
- x,y – данные (строки – переменные, колонки - наблюдения)
- rowvar – если не 0, то строка переменные, колонки – наблюдения (если 0, то наоборот)
- bias – определеяет нормировку, совместно с ddof
- mode - {‘valid’, ‘same’, ‘full’} – объем выводимых данных
- old_behavior – совместимость со старой версией (без комплексного сопряжения)
- bins – разбиение по интервалам
- range – массив границ по x и по y
- normed – нормированное
- minlength – минимальное число интервалов при выводе

Полиномы (numpy.polynomial)

Модуль полиномов обеспечивает стандартные функции работы с полиномами разного вида. В нем реализованы полиномы Чебышева, Лежандра, Эрмита, Лагерра. Для полиномов определены стандартные арифметические функции ‘+’, ‘-‘, ‘*’, ‘//’, деление по модулю, деление с остатком, возведение в степень и вычисление значения полинома. Важно задавать

область определения, т.к. часто свойства полинома (например, при интерполяции) сохраняются только на определенном интервале. В зависимости от класса полинома, сохраняются коэффициенты разложения по полиномам определенного типа, что позволяет получать разложение функций в ряд по полиномам разного типа.

Типы полиномов	**Описание**
Polynomial(coef[, domain, window])	разложение по степеням «x»
Chebyshev(coef[, domain, window])	разложение по полиномам Чебышева
Legendre(coef[, domain, window])	разложение по полиномам Лежандра
Hermite(coef[, domain, window])	разложение по полиномам Эрмита
HermiteE(coef[, domain, window])	разложение по полиномам Эрмита_E
Laguerre(coef[, domain, window])	разложение по полиномам Лагерра

- coef – массив коэффициентов в порядке увеличения
- domain – область определения проецируется на окно
- window – окно. Сдвигается и масштабируется до размера области определения

Некоторые функции (например, интерполяция данных) возвращают объект типа полином. У этого объекта есть набор методов, позволяющих извлекать и преобразовывать данные.

Методы полиномов	**Описание**
call__(z)	полином можно вызвать как функцию
convert([domain, kind, window])	конвертирует в полином другого типа, с другим окном и т.д
copy()	возвращает копию
cutdeg(deg)	обрезает полином до нужной степени
degree()	возвращает степень полинома
deriv([m])	вычисляет производную порядка m
fit(x, y, deg[, domain, rcond, full, w, window])	формирует полиномиальную интерполяцию степени deg для данных (x,y) по методу наименьших квадратов
fromroots(roots[, domain, window])	формирует полином по заданным корням
has_samecoef(p)	проверка на равенство коэффициентов.
has_samedomain(p)	проверка на равенство области определения
has_samewindow(p)	проверка на равенство окна
integ([m, k, lbnd])	интегрирование
linspace([n, domain])	возвращает x,y - значения на равномерной сетке по области определения
mapparms()	возвращает коэффициенты масштабирования
roots()	список корней
trim([tol])	создает полином с коэффициентами большими tol
truncate(size)	ограничивает ряд по количеству коеффициентов

- p – полином
- x, y – набор данных для аппроксимации
- deg – степень полинома
- domain – область определения
- rcond – относительное число обусловленности элементы матрицы интерполяции с собственными значениями меньшими данного будут отброшены.
- full – выдавать дополнительную информацию о качестве полинома
- w – веса точек
- window – окно

- roots – набор корней
- m – порядок производной (интеграла)
- k – константы интегрирования
- lbnd – нижняя граница интервала интегрирования
- n – число точек разбиения
- size – число ненулевых коэффициентов

Библиотека pandas. Структуры данных в pandas

Ядром pandas являются две структуры данных, в которых происходят все операции:

- Series
- Dataframes

`Series` — это структура, используемая для работы с последовательностью одномерных данных, а `Dataframe` — более сложная и подходит для нескольких измерений.

Однако особенности этих структур основаны на одной черте — интеграции в их структуру объектов `index` и `labels` (метки). С их помощью структурами становится очень легко манипулировать.

Series (серии)

`Series` — это объект библиотеки pandas, спроектированный для представления одномерных структур данных, похожих на массивы, но с дополнительными возможностями. Его структура проста, он состоит из двух связанных между собой массивов. Основной содержит данные (данные любого типа NumPy), а в дополнительном, `index`, хранятся метки.

Series	
index	value
0	12
1	-4
2	7
3	9

Создание объекта Series

Для создания объекта `Series` с предыдущего изображения необходимо вызвать конструктор `Series()` и передать в качестве аргумента массив, содержащий значения, которые необходимо включить.

```
>>> s = pd.Series([12,-4,7,9])
>>> s
0    12
1    -4
2     7
3     9
dtype: int64
```

Как можно увидеть по выводу, слева отображаются значения индексов, а справа — сами значения (данные).

Если не определить индекс при объявлении объекта, метки будут соответствовать индексам (положению в массиве) элементов объекта `Series`.

Однако лучше создавать `Series`, используя метки с неким смыслом, чтобы в будущем отделять и идентифицировать данные вне зависимости от того, в каком порядке они хранятся.
В таком случае необходимо будет при вызове конструктора включить параметр `index` и присвоить ему массив строк с метками.

```
>>> s = pd.Series([12,-4,7,9],
index=['a','b','c','d'])
>>> s
a    12
b    -4
c     7
d     9
dtype: int64
```

Если необходимо увидеть оба массива, из которых состоит структура, можно вызвать два атрибута: `index` и `values`.

```
>>>s.values
array([12, -4, 7, 9], dtype=int64)
>>>s.index
Index(['a', 'b', 'c', 'd'], dtype='object')
```

Выбор элементов по индексу или метке

Выбирать отдельные элементы можно по принципу обычных массивов numpy, используя для этого индекс.

```
>>> s[2]
7
```

Или же можно выбрать метку, соответствующую положению индекса.

```
>>> s['b']
-4
```

Таким же образом можно выбрать несколько элементов массива numpy с помощью следующей команды:

```
>>> s[0:2]
a    12
b    -4
dtype: int64
```

В этом случае можно использовать соответствующие метки, но указать их список в массиве.

```
>>> s[['b','c']]
b   -4
c    7
dtype: int64
```

Присваивание значений элементам

Понимая, как выбирать отдельные элементы, важно знать и то, как присваивать им новые значения.Можно делать это по индексу или по метке.

```
>>> s[1] = 0
>>> s
a    12
b     0
c     7
d     9
dtype: int64

>>> s['b'] = 1
>>> s
a    12
b     1
c     7
d     9
dtype: int64
```

Создание Series из массивов NumPy

Новый объект `Series` можно создать из массивов NumPy и уже существующих `Series`.

```
>>>arr = np.array([1,2,3,4])
>>> s3 = pd.Series(arr)
>>> s3
0    1
1    2
2    3
3    4
dtype: int32
```

```
>>> s4 = pd.Series(s)
>>> s4
a    12
b     1
c     7
d     9
dtype: int64
```

Важно запомнить, что значения в массиве NumPy или оригинальном объекте `Series` не копируются, а передаются по ссылке. Это значит, что элементы объекта вставляются динамически в новый `Series`. Если меняется оригинальный объект, то меняются и его значения в новом.

```
>>> s3
0    1
1    2
2    3
3    4
dtype: int32
```

```
>>>arr[2] = -2
>>> s3
0    1
1    2
2    -2
3    4
dtype: int32
```

На этом примере можно увидеть, что при изменении третьего элемента массива `arr`, меняется соответствующий элемент и в `s3`.

Фильтрация значений

Благодаря тому что основной библиотекой в pandas является NumPy, многие операции, применяемые к массивам NumPy, могут быть использованы и в случае с `Series`. Одна из таких — фильтрация значений в структуре данных с помощью условий.

Например, если нужно узнать, какие элементы в `Series` больше 8, то можно написать следующее:

```
>>> s[s > 8]
a    12
d     9
dtype: int64
```

Операции и математические функции

Другие операции, такие как операторы (`+`, `-`, `*` и `/`), а также математические функции, работающие с массивами NumPy, могут использоваться и для `Series`.

Для операторов можно написать простое арифметическое уравнение.

```
>>> s / 2
a    6.0
b    0.5
c    3.5
d    4.5
dtype: float64
```

Но в случае с математическими функциями NumPy необходимо указать функцию через `np`, а `Series` передать в качестве аргумента.

```
>>> np.log(s)
a    2.484907
b    0.000000
c    1.945910
d    2.197225
dtype: float64
```

Количество значений

В `Series` часто встречаются повторения значений. Поэтому важно иметь информацию, которая бы указывала на то, есть ли дубликаты или конкретное значение в объекте.

Так, можно объявить `Series`, в котором будут повторяющиеся значения.

```
>>>serd = pd.Series([1,0,2,1,2,3], index=['white','white','blue','green','
green','yellow'])
>>>serd
white     1
white     0
blue      2
green     1
green     2
yellow    3
dtype: int64
```

Чтобы узнать обо всех значениях в `Series`, не включая дубли, можно использовать функцию `unique()`. Возвращаемое значение — массив с уникальными значениями, необязательно в том же порядке.

```
>>>serd.unique()
array([1, 0, 2, 3], dtype=int64)
```

На `unique()` похожа функция `value_counts()`, которая возвращает не только уникальное значение, но и показывает, как часто элементы встречаются в `Series`.

```
>>>serd.value_counts()
2    2
1    2
3    1
0    1
dtype: int64
```

Наконец, isin() показывает, есть ли элементы на основе списка значений. Она возвращает булевые значения, которые очень полезны при фильтрации данных в Series или в колонке Dataframe.

```
>>>serd.isin([0,3])
white     False
white      True
blue      False
green     False
green     False
yellow     True
dtype: bool
>>>serd[serd.isin([0,3])]
white     0
yellow    3
dtype: int64
```

ЗначенияNaN

Как правило, NaN — это проблема, для которой нужно найти определенное решение, особенно при работе с анализом данных. Эти данные часто появляются при извлечении информации из непроверенных источников или когда в самом источнике недостает данных. Также значения NaN могут генерироваться в специальных случаях, например, при вычислении логарифмов для отрицательных значений, в случае исключений при вычислениях или при использовании функций. Есть разные стратегии работы со значениями NaN.

Несмотря на свою «проблемность» pandas позволяет явно определять NaN и добавлять это значение в структуры, например, в Series. Для этого внутри массива достаточно ввести np.NaN в том месте, где требуется определить недостающее значение.

```
>>> s2 = pd.Series([5,-3,np.NaN,14])
>>> s2
0     5.0
1    -3.0
2     NaN
3    14.0
dtype: float64
```

Функции isnull() и notnull() очень полезны для определения индексов без значения.

```
>>> s2.isnull()
0    False
1    False
2     True
3    False
dtype: bool
```

```
>>> s2.notnull()
0     True
1     True
2    False
3     True
dtype: bool
```

Они возвращают два объекта Series с булевыми значениями, где True указывает на наличие значение, а NaN — на его отсутствие. Функция isnull() возвращает True для значений NaN в Series, а notnull() — True в тех местах, где значение не равно NaN. Эти функции часто используются в фильтрах для создания условий.

```
>>> s2[s2.notnull()]
0     5.0
1    -3.0
3    14.0
dtype: float64
s2[s2.isnull()]
2   NaN
dtype: float64
```

DataFrame (датафрейм)

Dataframe — это табличная структура данных, напоминающая таблицы из MicrosoftExcel. Ее главная задача — позволить использовать многомерные Series. Dataframe состоит из упорядоченной коллекции колонок, каждая из которых содержит значение разных типов (числовое, строковое, булевое и так далее).

DataFrame

	columns		
index	color	object	price
0	blue	ball	1.2
1	green	pen	1.0
2	yellow	pencil	0.6
3	red	paper	0.9
4	white	mug	1.7

В отличие от Series у которого есть массив индексов с метками, ассоциированных с каждым из элементов, Dataframe имеет сразу два таких. Первый ассоциирован со строками (рядами) и напоминает таковой из Series. Каждая метка ассоциирована со всеми значениями в ряду. Второй содержит метки для каждой из колонок.

Dataframe можно воспринимать как dict, состоящий из Series, где ключи — названия колонок, а значения — объекты Series, которые формируют колонки самого объекта Dataframe. Наконец, все элементы в каждом объекте Series связаны в соответствии с массивом меток, называемым index.

Создание Dataframe

Создать датафрейм можно несколькими способами, например из словаря, значениями элементов которого являются объекты *Series*, а ключами - строки с названиями будующих столбцов:

```
# создаем словарь где ключ - это имя столбца,
# а значение - это объект Series:
data = {'first_column': pd.Series(range(5), index=list('edcba')),
        'second_column': pd.Series(range(5, 10), index=list('gfedc')),
        'third_column': pd.Series(range(2, 7), index=list('acdfg'))}

# создаемдатафрейм:
df = pd.DataFrame(data)
df
```

	first_column	second_column	third_column
a	4.0	NaN	2.0
b	3.0	NaN	NaN
c	2.0	9.0	3.0
d	1.0	8.0	4.0
e	0.0	7.0	NaN
f	NaN	6.0	5.0
g	NaN	5.0	6.0

Можно указать необходимые индексы и столбцы:

```
pd.DataFrame(data,
             # указываем нужные индексы:
index=['c', 'd'])
```

	first_column	second_column	third_column
c	2	5	3
d	3	6	4

```
pd.DataFrame(data,
             # указываем нужные индексы:
index=['c', 'd', 'f', 'g', 'z'],
```

```
# указываемнеобходимыестолбцы:
columns=['second_column',
'third_column',
                      'fourth_column'])
```

	second_column	third_column	fourth_column
c	5.0	3.0	NaN
d	6.0	4.0	NaN
f	8.0	5.0	NaN
g	9.0	6.0	NaN
z	NaN	NaN	NaN

Датафрейм может быть создан из структурированного массива NumPy:

```
data = np.array([('a', 11, .9),
                 ('b', 22, .8),
                 ('c', 33, 0.7)],
dtype=[('col_1', 'U1'),
                       ('col_2', 'i2'),
                       ('col_3', 'f2')])

pd.DataFrame(data)
```

	col_1	col_2	col_3
0	a	11	0.899902
1	b	22	0.799805
2	c	33	0.700195

```
pd.DataFrame(data, index=['one', 'two', 'three'])
```

	col_1	col_2	col_3
one	a	11	0.899902
two	b	22	0.799805
three	c	33	0.700195

```
pd.DataFrame(data,
             index=['one', 'two', 'three'],
             columns=['col_2', 'col_1', 'col_4'])
```

	col_2	col_1	col_4
one	11	a	NaN
two	22	b	NaN
three	33	c	NaN

Выбор элементов

Если нужно узнать названия всех колонок `Dataframe`, можно вызвать атрибут `columns` для экземпляра объекта.

Весь набор данных можно получить с помощью атрибута `values`.

Указав в квадратных скобках название колонки, можно получить значений в ней.

Возвращаемое значение — объект `Series`. Название колонки можно использовать и в качестве атрибута.

Для строк внутри `Dataframe` используется атрибут `loc` со значением индекса нужной строки.

Возвращаемый объект — это снова `Series`, где названия колонок — это уже метки массива индексов, а значения — данные `Series`.

Для выбора нескольких строк можно указать массив с их последовательностью.

Если необходимо извлечь часть `Dataframe` с конкретными строками, для этого можно использовать номера индексов. Она выведет данные из соответствующей строки и названия колонок.

Возвращаемое значение — объект `Dataframe` с одной строкой. Если нужно больше одной строки, необходимо просто указать диапазон.

Наконец, если необходимо получить одно значение из объекта, сперва нужно указать название колонки, а потом — индекс или метку строки.

Вхождение значений

Функция `isin()` используется с объектами `Series` для определения вхождения значений в колонку. Она же подходит и для объектов `Dataframe`.

Возвращается `Dataframe` с булевыми значениями, где `True` указывает на те значения, где членство подтверждено. Если передать это значение в виде условия, тогда вернется `Dataframe`, где будут только значения, удовлетворяющие условию.

Удаление колонки

Для удаления целой колонки и всего ее содержимого используется команда `del`.

Методы описательной статистики

В Pandas существует множества функций для вычисления описательных статистик данных, хранящихся в сериях и датафреймах. Многие из этих функций являются агрегирующими и выдают результат в виде серий и датафреймов меньших размеров (например `sum()`, `mean()`), другие (например `cumsum()` и `cumprod()`) и выдают результат того же размера.

Все эти методы принимают аргумент `axis` который позволяет задать ось, вдоль которой должна выполняться функция. Для серий данный аргумент не имеет никакого смысла, так как они являются одномерными, а для датафреймов он может быть равен одному из следующих значений:

- `'index'` (т.е. `axis = 'index'` или что тоже самое `axis = 0`) - что соответствует вычислениям по столбцам (данное значение установлено по умолчанию);
- `'columns'` (т.е. `axis = 'columns'` или что тоже самое `axis = 1`)- соответствует вычислениям вдоль строк.

```
s =pd.Series(range(10))
s.sum(),s.mean()
(45, 4.5)
df =pd.DataFrame(np.arange(16,dtype=np.float).reshape(4,4),
                 index=list('abcd'),
                 columns=['col_'+str(i)foriinrange(4)])
df
```

ol_0	ol_1	ol_2	ol_3
.0	.0	.0	.0
.0	.0	.0	.0
.0	.0	0.0	1.0
2.0	3.0	4.0	5.0

```
# сумма по столбцам:
df.sum()# axis='index' по умолчанию
col_0    24
```

```
# сумма по строкам:
df.sum(axis='columns')
a     6
```

```
col_1    28
col_2    32
col_3    36
dtype: int64
```

```
b    22
c    38
d    54
dtype: int64
```

У всех этих методов есть аргумент `skipna` позволяющий исключать отсутствующие значения:

```
df['col_3'][0]=np.nan
df
```

ol_0	ol_1	ol_2	ol_3
.0	.0	.0	aN
.0	.0	.0	.0
.0	.0	0.0	1.0
2.0	3.0	4.0	5.0

```
df.sum(axis=1)
a     3.0
b    22.0
c    38.0
d    54.0
dtype: float64
```

```
df.sum(axis=1,skipna=False)
a     NaN
b    22.0
c    38.0
d    54.0
dtype: float64
```

Учитывая, что арифметические операции транслируются, то данные методы могут быть использованы в математических выражениях:

```
((df -df.mean())/df.std()).std()
col_0    1.0
col_1    1.0
col_2    1.0
col_3    1.0
dtype: float64
```

Аккумулирующие функции сохраняют размеры датафреймов и серий, но если в них содержатся NaN-ы то их расположение так же сохраняется:

```
pd.Series([1,2,np.nan,4,5,np.nan,6]).cumsum()
0     1.0
1     3.0
2     NaN
3     7.0
4    12.0
5     NaN
6    18.0
dtype: float64
df.cumsum()
```

ol_0	ol_1	ol_2	ol_3
.0	.0	.0	aN
.0	.0	.0	.0
2.0	5.0	8.0	8.0
4.0	8.0	2.0	3.0

Так же обратите внимание на то, что некоторые аналогичные методы NumPy могут игнорировать значения NaN в сериях:

```
np.sum(df['col_3'])
33.0
np.sum(df['col_3'].to_numpy())
nan
np.nansum(df['col_3'].to_numpy())
33.0
```

Вот краткая информация по основным функция описательной статистики:

- `count` - количество значений не равных NaN;;
- `sum` - сумма значений элементов;
- `mean` - среднее значение элементов;;
- `mad` - среднее абсолютных значений элементов;;
- `median` - медиана значений элементов (половина значений меньше медианы, другая половина больше);;
- `min` - минимальное значение;;
- `max` - максимальное значение;;
- `mode` - наиболее часто-встречающееся (типичное) значение;
- `abs` - абсолютное значение элементов;;
- `prod` - произведение значений;
- `std` - стандартное отклонение с поправкой Бесселя (используется n - 1 вместо n);
- `var` - несмещенная (исправленная) дисперсия;
- `sem` - стандартная ошибка среднего;
- `skew` - асимметрия распределения выборки (третий момент);
- `kurt` - куртозис (коэфициентэксцеса) распределения выборки, характеризует остроту вершин распределения выборки и тяжесть (толщину) хвостов;
- `quantile` - квантиль значений, т.е. значение которое не будет превышено с заданной вероятностью (т.е. просто процент количества значений, которые являются меньше чем указанное значение);
- `cumsum` - кумулятивная сумма;
- `cumprod` - кумулятивное произведение;
- `cummax` - кумулятивный максимум;
- `cummin` - кумулятивный минимум;

Краткая статистическая сводка

Иногда бывает полезно быстро узнать какие-нибудь сводные статистики о данных в серии или каждого столбца в датафрейме. Для этого существует метод `describe()` который вычисляет эти статистики (без учета значений NaN):

```
s =pd.Series(np.random.randn(1000))
s[::3]=np.nan
s
0           NaN
1      0.694777
2     -0.260088
3           NaN
4     -0.067500
         ...
995   -0.058663
996         NaN
997    0.995671
998    0.298379
999         NaN
Length: 1000, dtype: float64
```

```
s.describe()
count    666.000000
mean      -0.063356
std        0.980980
min       -3.485490
25%       -0.695968
50%       -0.058447
75%        0.598769
max        2.737608
dtype: float64
```

Глядя на данный вывод, мы можем заметить, что среднее значение близко к 0, а стандартное значение близко к 1. К тому же 50-й процентиль (он же второй квантиль, он же

медиана) тоже очень близок к среднему значению, так что перед нами, судя по всему, действительно, нормально распределенная случайная величина.

Для датафреймов подобные статистики вычисляются для каждого столбца в отдельности:

```
data =pd.DataFrame(np.random.randn(2000,4),
                   columns =['col_'+str(i)foriinrange(4)])
data.iloc[::3]=np.nan
data
```

	co l_0	co l_1	co l_2	co l_3
	Na N	Na N	Na N	Na N
	0. 673580	0. 469629	2. 033159	- 1.201297
	1. 347816	- 0.652112	0. 238076	0. 692468
	Na N	Na N	Na N	Na N
	1. 637430	- 0.292464	0. 160859	- 0.463325
	.. ..	.. .	.. .	.. .
	Na N	Na N	Na N	Na N
995	0. 975141	- 0.754659	0. 769076	1. 164425
996	- 0.644748	1. 060235	0. 949261	- 0.667111
997	Na N	Na N	Na N	Na N
998	- 2.609587	- 1.201645	- 1.526409	0. 419282
999				

```
data.describe()
```

	col_ 0	col_ 1	col_ 2	col_ 3
ount	1333 .000000	1333 .000000	1333 .000000	1333 .000000
ean	- 0.007838	- 0.054281	0.00 7605	0.00 5540
td	0.99 2171	1.02 0226	0.98 1725	1.02 0494
in	- 3.729444	- 3.570644	- 3.632059	- 2.835269
5%	- 0.652890	- 0.778349	- 0.677309	- 0.665894
0%	0.02 8582	- 0.055274	0.01 4708	0.04 2450
5%	0.67 3580	0.64 3916	0.69 0836	0.67 4617
ax	3.29 0922	3.80 3188	3.00 8416	3.39 0709

Для серий состоящих из нечисловых значений метод `describe()` возвращает:

- `count` - количество не равных NaN элементов;
- `unique` - количество уникальных элементов;
- `top` - наиболее часто встречающееся значение;
- `freq` - число появлений `top`-значения в данных.

```
s =pd.Series(list('ababaaacbabbaa'))
s[[1,3,5]]=np.nan
```

```
s.describe()
count     11
```

```
s
0       a
1     NaN
2       a
3     NaN
4       a
5     NaN
6       a
7       c
8       b
9       a
10      b
11      b
12      a
13      a
dtype: object
```

```
unique     3
top        a
freq       7
dtype: object
```

Для датафреймов, состоящих из числовых и нечисловых столбцов, по умолчанию, всегда выводится сводка только по числовым столбцам.

Указать, какие именно столбцы должны попасть в сводку можно с помощью параметра `include`.

Метод `describe()` позволяет так же указывать какого типа столбцы должны попасть в сводку, но обычно это очень редко используется на практике. Однако, если практика на то и практика что бы там могло понадобиться все что угодно, так что не забывайте обращаться к официальной документации.

Очень часто бывает полезным подсчитать количество уникальных значений в серии (кроме NaN, естественно), для этого подойдет метод `nunique()`.

Индексы минимальных и максимальных значений

Что бы узнать индекс максимального или минимального элемента в серии можно воспользоваться методом `idxmin()` или `idxmax()` (если таких значений несколько то будет возвращен индекс первого встретившегося):

```
s =pd.Series([1,1,2,2,3,3])
s.idxmin(),s.idxmax()
(0, 4)
s
=pd.Series(np.random.randint(100,1000,1000))
```

```
# индекс элемента с максимальным значением
s.idxmax()
980
# сам элемент с максимальным значением
s[s.idxmax()]
999
```

Подсчет значений и мода значений

Иногда данные являются категориальными или состоят из небольшого множества значений. В таких ситуациях часто возникает необходимость, подсчета количества каждого из значений. Сделать это можно с помощью метода `value_counts()`:

```
s
=pd.Series(list('ababccaabccab'))
s
0     a
1     b
2     a
3     b
4     c
5     c
6     a
7     a
8     b
9     c
10    c
```

```
s
=pd.Series(np.random.randint(0,10,2000))
s.value_counts()
8    222
0    214
3    213
4    209
7    206
9    204
5    196
1    192
2    179
6    165
dtype: int64
```

```
11    a
12    b
dtype: object
s.value_counts()
a    5
b    4
c    4
dtype: int64
```

Данный метод так же доступен в виде функции и может применяться к обычным одномерным массивам:

```
data =np.random.randint(0,10,2000)

pd.value_counts(data)
7    214
6    212
5    207
2    207
8    202
0    200
1    193
4    191
9    187
3    187
dtype: int64
```

Как видите, подсчет возвращается в неотсортированном виде, что бы исправить это можно воспользоваться методом sort_index() (или воспользоваться дополнительными параметра данного метода):

```
pd.value_counts(data).sort_index()
0    200
1    193
2    207
3    187
4    191
5    207
6    212
7    214
8    202
9    187
dtype: int64
```

Метод `value_counts()` может использоваться и для подсчета уникальных комбинаций в столбцах датафреймов:

```
data =np.random.randint(0,2,20).reshape(10,2)
df =pd.DataFrame(data, columns=['a','b'])
df
```

```
df.value_counts()
a  b
0  1    6
0    2
1  1    1
   0    1
dtype: int64
```

Получить наиболее часто встречающееся значение (или значения, если их несколько) можно с помощью метода `mode()`:

```
s =pd.Series([1,2,3,3,3,4,5])
s.mode()
0    3
dtype: int64
```

```
s =pd.Series([1,1,1,2,3,3,3])
s.mode()
0    1
1    3
dtype: int64
```

```
data_1 =np.random.randint(0,5,100)
data_2 =np.random.randint(5,10,100)
df =pd.DataFrame({'a': data_1,
'b': data_2})

df.mode()
```

```
.0

aN
```

Данный вывод означает что в столбце **a** чаще всего встречается значение 2, а в столбце **b** чаще всего встречаются значения 6 и 8 которых поровну в чем очень дегко убедиться:

```
df['a'].value_counts()
2    26
4    24
0    19
1    16
3    15
Name: a, dtype: int64
```

```
df['b'].value_counts()
8    24
6    24
7    22
9    16
5    14
Name: b, dtype: int64
```

Дискретизация непрерывных значений

Некоторые величины, например такие как возраст или количество выпавших осадков, являются непрерывными. Тем не менее иногда приходится раскладывать наблюдения таких величин по дискретным "ящикам" - интервалам значений. Для возраста, мы можем выделить интервалы (18, 35], (35, 75], (75, 110] и назвать их "молодой", "зрелый" и "пожилой" соответственно. Выполнить подобное разбиение непрерывной величины в Pandas можно с помощью метода `cut()`:

```
s =pd.Series([0.11,0.19,0.23,0.27,0.33,0.39])

hist_s=pd.cut(s,3)
hist_s
0     (0.11, 0.203]
1     (0.11, 0.203]
2    (0.203, 0.297]
3    (0.203, 0.297]
4     (0.297, 0.39]
5     (0.297, 0.39]
dtype: category
Categories (3, interval[float64]): [(0.11, 0.203] < (0.203, 0.297] <
(0.297, 0.39]]
```

В данном случае, мы разбили значения `[0.11, 0.19, 0.23, 0.27, 0.33, 0.39]` на три непересекающихся интервала одинаковой длины, причем в каждом оказалось по два значения, в чем легко убедиться:

```
pd.value_counts(hist_s)
(0.297, 0.39]     2
(0.203, 0.297]    2
(0.11, 0.203]     2
dtype: int64
```

Интервалы вовсе не обязательно должны быть одинаковыми и вместо их количества мы можем задавать их границы:

```
data =10*np.random.randn(300)

hist_data=pd.cut(data,[-30,-5,0,5,30])

hist_data
```

```
[(0, 5], (-30, -5], (5, 30], (-5, 0], (0, 5], ..., (-30, -5], (5, 30], (-30, -5], (-30, -5], (5, 30]]
Length: 300
Categories (4, interval[int64]): [(-30, -5] < (-5, 0] < (0, 5] < (5, 30]]
pd.value_counts(hist_data)
(-30, -5]    94
(5, 30]      88
(-5, 0]      65
(0, 5]       51
dtype: int64
```

Вообще, данные могут быть и целого типа, но тогда нужно указывать вещественные границы интервалов, так чтобы в них точно попадали все значения:

```
data =np.random.randint(0,10,300)
hist_data=pd.cut(data,[0,5,10])
hist_data
[(0, 5], (5, 10], (0, 5], (5, 10], (5, 10], ..., NaN, (5.0, 10.0], (0.0, 5.0], (0.0, 5.0], (0.0, 5.0]]
Length: 300
Categories (2, interval[int64]): [(0, 5] < (5, 10]]
```

В примере выше мы указали границы, как [0, 5, 10] что соответствует двум интервалам (0, 5] и (5, 10], но 0 не входит в интервал (0, 5], а значит не учитывается при дискретизации, в чем тоже очень легко убедиться:

```
pd.value_counts(hist_data).sum()
272
```

При указывании границ интервалов можно использовать значения np.inf, т.е. по сути можно просто разбить данные пополам относительно некоторого значения:

```
data =np.random.randn(300)
hist_data=pd.cut(data,[-np.inf,0, np.inf])
pd.value_counts(hist_data)
(-inf, 0.0]    155
(0.0, inf]     145
dtype: int64
```

Еще один способ дискретизации заключается в том что бы указывать не количество интервалов или их границы, а квантили. Сделать это можно с помощью метода `qcut()`:

```
data=np.random.randn(50)

hist_data=pd.qcut(data,[0,0.25,0.75,1])

hist_data
[(0.411, 1.765], (-0.715, 0.411], (-2.1229999999999998, -0.715], (0.411, 1.765], (-0.715, 0.411], ..., (0.411, 1.765], (-0.715, 0.411], (-0.715, 0.411], (-0.715, 0.411]]
Length: 50
Categories (3, interval[float64]): [(-2.1229999999999998, -0.715] < (-0.715, 0.411] < (0.411, 1.765]]
pd.value_counts(hist_data)
(-0.715, 0.411]                  24
(0.411, 1.765]                   13
(-2.1229999999999998, -0.715]    13
dtype: int64
```

Метод `qcut()` полезен когда мы работаем с асемметричным распределением:

```
data =np.random.beta(2,3,150)

hist_data=pd.qcut(data,[0,0.25,0.75,1])

hist_data
[(0.535, 0.919], (0.0114, 0.24], (0.535, 0.919], (0.24, 0.535], (0.0114, 0.24], ..., (0.24, 0.535], (0.0114, 0.24], (0.24, 0.535], (0.0114, 0.24], (0.24, 0.535]]
Length: 150
```

```
Categories (3, interval[float64]): [(0.0114, 0.24] < (0.24, 0.535] < (0.535, 0.919]]
pd.value_counts(hist_data)
(0.24, 0.535]     74
(0.535, 0.919]    38
(0.0114, 0.24]    38
dtype: int64
```

В данном случае мы просто получаем интервалы такой ширины, что значения выборки могут попасть в них только с указанной межквантильной вероятностью. Например, для интервала границы которого определяют 0.25 и 0.75 квантили, вероятность попадания значений в него равна 0.5. А для остальных двух интервалов вероятность равна 0.25.

Ну и конечно же удобен такой метод тем, что значения всегда можно очень легко разделить на две части используя медиану (0.5-й квантиль) в качестве разделителя:

```
data =np.random.beta(2,3,150)

hist_data=pd.qcut(data,[0,0.5,1])

hist_data
[(0.395, 0.863], (0.00799, 0.395], (0.00799, 0.395], (0.00799, 0.395], (0.00799, 0.395], ..., (0.395, 0.863], (0.00799, 0.395], (0.00799, 0.395], (0.395, 0.863], (0.00799, 0.395]]
Length: 150
Categories (2, interval[float64]): [(0.00799, 0.395] < (0.395, 0.863]]
pd.value_counts(hist_data)
(0.395, 0.863]      75
(0.00799, 0.395]    75
dtype: int64
```

Типы графиков в matplotlib

Поскольку визуализация — основная цель библиотеки `matplotlib` , то этот раздел является очень важным. Умение выбрать правильный тип графика является фундаментальным навыком, ведь неправильная репрезентация может привести к тому, что данные, полученные в результате качественного анализа данных, будет интерпретированы неверно.

Для выполнения кода импортируйте `pyplot` и `numpy`

```
import matplotlib.pyplot as plt
import numpy as np
```

Линейные графики

Линейные графики являются самыми простыми из всех. Такой график — это последовательность точек данных на линии. Каждая точка состоит из пары значений (x, y), которые перенесены на график в соответствии с масштабами осей (x и y).

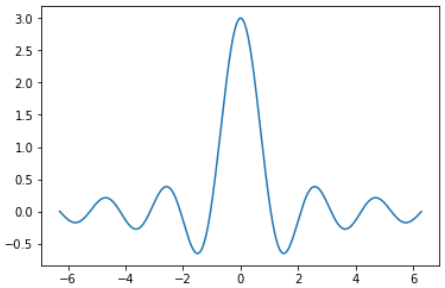
В качестве примера можно вывести точки, сгенерированные математической функцией. Пример:

```
y = sin (3 * x) / x
```

Для создания последовательности точек данных нужно создать два массива `NumPy`. Сначала создадим массив со значениями x для оси x. Для определения последовательности увеличивающихся значений используем функцию `np.arrange()`. Поскольку функция синусоидальная, то значениями должны быть числа кратные π (`np.pi`). Затем с помощью этой последовательности можно получить значения y, применив для них функцию `np.sin()`.

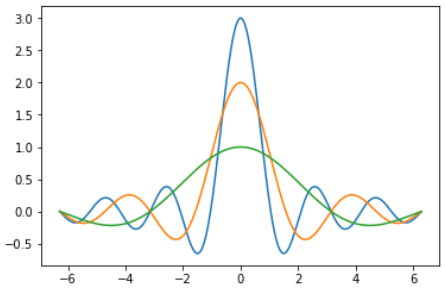
После этого остается лишь вывести все точки на график с помощью функции `plot()`. Результатом будет линейный график.

```
x =np.arange(-2*np.pi,2*np.pi,0.01)
y =np.sin(3*x)/x
plt.plot(x,y)
plt.show()
```



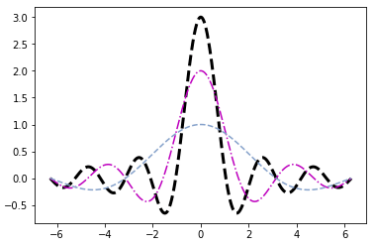
Этот пример можно расширить для демонстрации семейства функций, например, такого (с разными значениями `n`):

```
x =np.arange(-2*np.pi,2*np.pi,0.01)
y =np.sin(3*x)/x
y2 =np.sin(2*x)/x
y3 =np.sin(x)/x
plt.plot(x,y)
plt.plot(x,y2)
plt.plot(x,y3)
plt.show()
```



Каждой линии автоматически присваивается свой цвет. При этом все графики представлены в одном масштабе. Это значит, что точки данных связаны с одними и теми же осями x и y. Вот почему каждый вызов функции `plot()` учитывает предыдущие вызовы, так что объект `Figure` применяет изменения с учетом прошлых команд еще до вывода (для вывода используется `show()`).

```
x =np.arange(-2*np.pi,2*np.pi,0.01)
y =np.sin(3*x)/x
y2 =np.sin(2*x)/x
y3 =np.sin(x)/x
plt.plot(x,y,'k--',linewidth=3)
plt.plot(x,y2,'m-.')
plt.plot(x,y3,color='#87a3cc',linestyle='--')
plt.show()
```

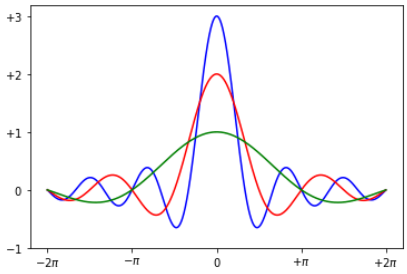


Вне зависимости от настроек по умолчанию можно выбрать тип начертания, цвет и так далее. Третьим аргументом функции `plot()` можно указать коды цветов, типы линий и все этой в одной строке. Также можно использовать два именованных аргумента отдельно: `color` — для цвета и `linestyle` — для типа линии.

Код	Цвет		Код	Цвет
b	голубой		m	пурпурный
g	зеленый		y	желтый
r	красный		k	черный
c	сине-зеленый		w	белый

На графике определен диапазон от — 2π до 2π на оси x, но по умолчанию деления обозначены в числовой форме. Поэтому их нужно заменить на множители числа π. Также можно поменять делители на оси y. Для этого используются функции `xticks()` и `yticks()`. Им нужно передать список значений. Первый список содержит значения, соответствующие позициям, где деления будут находиться, а второй — их метки. В этом случае будут использоваться LaTeX-выражения, что нужно для корректного отображения π. Важно не забыть добавить знаки `$` в начале и конце, а также символ `r` в качестве префикса.

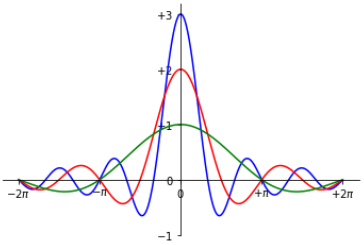
```
x =np.arange(-2*np.pi,2*np.pi,0.01)
y =np.sin(3*x)/x
y2 =np.sin(2*x)/x
y3 =np.sin(x)/x
plt.plot(x,y,color='b')
plt.plot(x,y2,color='r')
plt.plot(x,y3,color='g')
plt.xticks([-2*np.pi,-np.pi,0,np.pi,2*np.pi],
[r'$-2\pi$',r'$-\pi$',r'$0$',r'$+\pi$',r'$+2\pi$'])
plt.yticks([-1,0,1,2,3],
[r'$-1$',r'$0$',r'$+1$',r'$+2$',r'$+3$'])
plt.show()
```



Пока что на всех рассмотренных графиках оси x и y изображались на краях объекта `Figure` (по границе рамки). Но их же можно провести так, чтобы они пересекались — то есть, получит декартову система координат.

Для этого нужно сперва получить объект `Axes` с помощью функцию `gca`. Затем с его помощью можно выбрать любую из четырех сторон, создав область с границами и определив положение каждой: справа, слева, сверху и снизу. Ненужные части обрезаются (справа и снизу), а с помощью функции `set_color()` задается значение `none`. Затем стороны, которые соответствуют осям x и y, проходят через начало координат (0, 0) с помощью функции `set_position()`.

```
x =np.arange(-2*np.pi,2*np.pi,0.01)
y =np.sin(3*x)/x
y2 =np.sin(2*x)/x
y3 =np.sin(x)/x
plt.plot(x,y,color='b')
plt.plot(x,y2,color='r')
plt.plot(x,y3,color='g')
plt.xticks([-2*np.pi,-np.pi,0,np.pi,2*np.pi],
[r'$-2\pi$',r'$-\pi$',r'$0$',r'$+\pi$',r'$+2\pi$'])
plt.yticks([-1,0,1,2,3],
[r'$-1$',r'$0$',r'$+1$',r'$+2$',r'$+3$'])
ax =plt.gca()
```



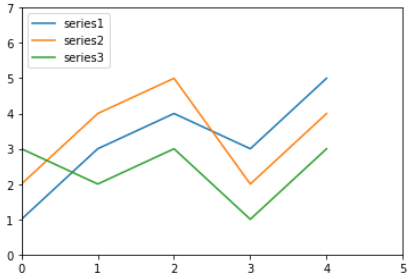
```
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
plt.show()
```

Теперь график будет состоять из двух пересекающихся в центре осей, который представляет собой начало декартовой системы координат.

Линейные графики с `pandas`

Рассмотрим более практический и приближенный к анализу данных пример. С ним будет видно, насколько просто использовать библиотеку `matplotlib` для объектов `Dataframe` из библиотеки `pandas`. Визуализация данных в виде линейного графика — максимально простая задача. Достаточно передать объект в качестве аргумента функции `plot()` для получения графика с несколькими линиями.

```
import pandas as pd
data ={'series1':[1,3,4,3,5],
'series2':[2,4,5,2,4],
'series3':[3,2,3,1,3]}
df =pd.DataFrame(data)
x =np.arange(5)
plt.axis([0,5,0,7])
plt.plot(x,df)
plt.legend(data, loc=2)
plt.show()
```



Гистограммы

Гистограмма состоит из примыкающих прямоугольников, расположенных вдоль оси x, которые разбиты на дискретные интервалы, их называют `bins`. Их площадь пропорциональна частоте конкретного интервала. Такой способ визуализации часто используют в статистике для демонстрации распределения.

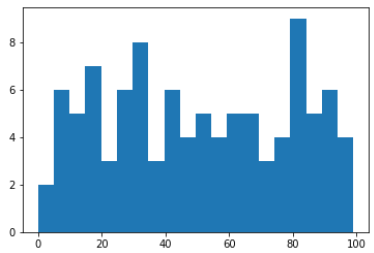
Для представления гистограммы в pyplot есть функция `hist()`. У нее также есть особенности, которых не найти у других функций, отвечающих за создание графиков. `hist()` не только рисует гистограмму, но также возвращает кортеж значений, представляющих собой результат вычислений гистограммы. Функция `hist()` может реализовывать вычисление гистограммы, чего достаточно для предоставления набора значений и количества интервалов, на которых их нужно разбить. Наконец `hist()` отвечает за разделение интервала на множество и вычисление частоты каждого. Результат этой операции не только выводится в графической форме, но и возвращается в виде кортежа.

Для понимания операции лучше всего воспользоваться практическим примером. Сгенерируем набор из 100 случайных чисел от 0 до 100 с помощью `random.randint()`.

```
pop =np.random.randint(0,100,100)
pop
array([33, 90, 10, 68, 18, 67,  6, 54, 32, 25, 90,  6, 48, 34, 59, 70, 37,
50, 86,  7, 49, 40, 54, 94, 95, 20, 83, 59, 33,  0, 81, 18, 26, 69,
        2, 42, 51,  7, 42, 90, 94, 63, 14, 14, 71, 25, 85, 99, 40, 62, 29,
       42, 27, 98, 30, 89, 21, 78, 17, 33, 63, 80, 61, 50, 79, 38, 96,  8,
       85, 19, 76, 32, 19, 14, 37, 62, 24, 30, 19, 80, 55,  5, 94, 74, 85,
       59, 65, 17, 80, 11, 81, 84, 81, 46, 82, 66, 46, 78, 29, 40])
```

Дальше создаем гистограмму из этих данных, передавая аргумент функции `hist()`. Например, нужно разделить данные на 20 интервалов (значение по умолчанию — 10 интервалов). Для этого используется именованный аргумент `bin`.

```
n,bin, patches =plt.hist(pop, bins=20)
plt.show()
```



Гистограмма позволяет исследователю:

- оценить ряд статистических показателей,
- сделать выводы о функции распределения,
- определить возможные отклонения,
- сравнить два набора данных (в частности, результаты до и после произведенных действий или внедрения проекта).

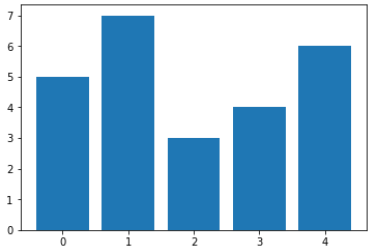
Величины, которые можно оценить на гистограмме:

- распределение наблюдений (distribution): визуальная оценка, на какое из известных распределений похожа форма графика;
- наибольшую концентрацию данных – моду (mode): наличие 2 или долее мод указывает на присутствии специальных факторов, влияющих на исследуемую систему или процесс;
- минимальное и максимальное значения (min и max);
- размах (range);
- степень асимметрии – скос (skewness): симметричный или ассиметричный (отрицательная – левый хвост или положительная - правый);
- эксцесс (kurtosis) - числовая характеристика степени остроты пика;
- наличие явных выбросов (outliers);
- возможное присутствие нескольких распределений (популяций);
- ширину интервалов – дистанцию между правым и левым краями частотной ячейки по оси X;
- количество интервалов – общее (в том числе и нулевые значения) количество частотных ячеек гистограммы.

Столбчатые диаграммы

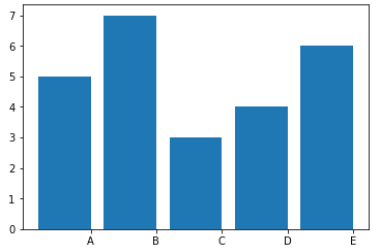
Еще один распространенный тип графиков — столбчатые диаграммы. Они похожа на гистограммы, но на оси x тут располагаются не числовые значения, а категории. В matplotlib для реализации столбчатых диаграмм используется функция `bin()`.

```
index =[0,1,2,3,4]
values =[5,7,3,4,6]
plt.bar(index,values)
plt.show()
```



На последней диаграмме видно, что метки на оси x написаны под каждым столбцом. Поскольку каждый из них относится к отдельной категории, правильнее обозначать их строками. Для этого используется функция `xticks()`. А для правильного размещения нужно передать список со значениями позиций в качестве первого аргумента в той же функции. Результатом будет такая диаграмма.

```
index =np.arange(5)
values1 =[5,7,3,4,6]
plt.bar(index, values1)
plt.xticks(index+0.4,['A','B','C','D','E'])
plt.show()
```



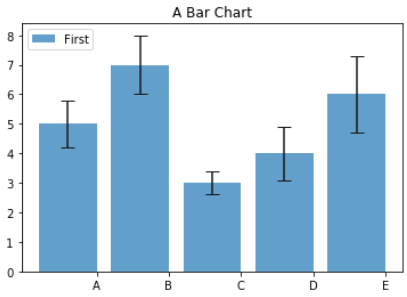
Есть и множество других операций, которые можно выполнить для улучшения диаграммы. Каждая из них выполняется за счет добавления конкретного именованного аргумента в `bar()`. Например, можно добавить величины стандартного отклонения с помощью аргумента `yerr` вместе с соответствующими значениями. Часто этот аргумент используется вместе с `error_kw`, который принимает друге аргументы, отвечающие за представление погрешностей. Два из них — это `eColor`, который определяет цвета колонок погрешностей и `capsize` — ширину поперечных линий, обозначающих окончания этих колонок.

Еще один именованный аргумент — `alpha`. Он определяет степень прозрачности цветной колонки. Его значением может быть число от 0 до 1, где 0 — полностью прозрачный объект.

Также крайне рекомендуется использовать легенду, за которую отвечает аргумент `label`.

Результат — следующая столбчатая диаграмма с колонками погрешностей.

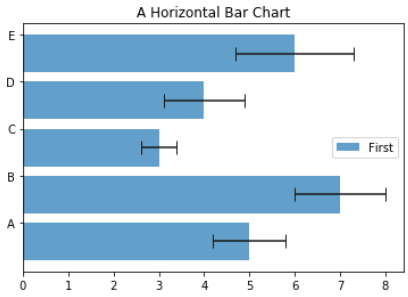
```
index =np.arange(5)
values1 =[5,7,3,4,6]
std1 =[0.8,1,0.4,0.9,1.3]
plt.title('A Bar Chart')
plt.bar(index, values1,yerr=std1,
error_kw={'ecolor':'0.1','capsize':6},
alpha=0.7,label='First')
plt.xticks(index+0.4,['A','B','C','D','E'])
plt.legend(loc=2)
plt.show()
```



Горизонтальные столбчатые диаграммы

В предыдущем разделе столбчатая диаграмма была вертикальной. Но блоки могут располагаться и горизонтально. Для этого режима есть специальная функция `barh()`. Аргументы и именованные аргументы, которые использовались для `bar()` будут работать и здесь. Единственное изменение в том, что поменялись роли осей. Категории теперь представлены на оси y, а числовые значения — на x.

```
index =np.arange(5)
values1 =[5,7,3,4,6]
std1 =[0.8,1,0.4,0.9,1.3]
plt.title('A Horizontal Bar Chart')
plt.barh(index, values1,xerr=std1,
error_kw={'ecolor':'0.1','capsize':6},
alpha=0.7,label='First')
plt.yticks(index+0.4,['A','B','C','D','E']
)
plt.legend(loc=5)
plt.show()
```



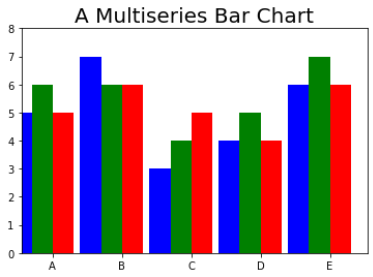
Многорядные столбчатые диаграммы

Как и линейные графики, столбчатые диаграммы широко используются для одновременного отображения больших наборов данных. Но в случае с многорядными работает особая структура. До сих пор во всех примерах определялись последовательности индексов, каждый из которых соответствует столбцу, относящемуся к оси x. Индексы

представляют собой и категории. В таком случае столбцов, которые относятся к одной и той же категории, даже больше.

Один из способов решения этой проблемы — разделение пространства индекса (для удобства его ширина равна 1) на то количество столбцов, которые к нему относятся. Также рекомендуется добавлять пустое пространство, которое будет выступать пропусками между категориями.

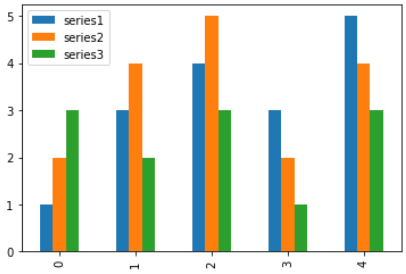
```
index =np.arange(5)
values1 =[5,7,3,4,6]
values2 =[6,6,4,5,7]
values3 =[5,6,5,4,6]
bw=0.3
plt.axis([0,5,0,8])
plt.title('A Multiseries Bar Chart',fontsize=20)
plt.bar(index, values1,bw, color='b')
plt.bar(index+bw, values2,bw, color='g')
plt.bar(index+2*bw, values3,bw, color='r')
plt.xticks(index+1.5*bw,['A','B','C','D','E'])
plt.show()
```



Многорядные столбчатые диаграммы с Dataframe из pandas

Как и в случае с линейными графиками matplotlib предоставляет возможность представлять объекты Dataframe с результатами анализа данных в форме столбчатых графиков. В этом случае все происходит даже быстрее и проще. Нужно лишь использовать функцию `plot()` по отношению к объекту Dataframe и указать внутри именованный аргумент `kind`, ему требуется присвоить тип графика, который будет выводиться. В данном случае это `bar`. Без дополнительных настроек результат должен выглядеть как на следующем изображении.

```
importpandasas pd
index =np.arange(5)
data ={'series1':[1,3,4,3,5],
'series2':[2,4,5,2,4],
'series3':[3,2,3,1,3]}
df =pd.DataFrame(data)
df.plot(kind='bar')
plt.show()
```

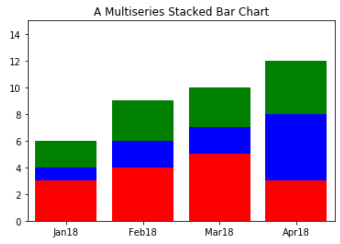


Многорядные сложенные столбчатые графики

Еще один способ представления многорядного столбчатого графика — сложенная форма, где каждый столбец установлен поверх другого. Это особенно полезно в том случае, когда нужно показать общее значение суммы всех столбцов.

Для превращения обычного многорядного столбчатого графика в сложенный нужно добавить именованный аргумент `bottom` в каждую функцию `bar()`. Каждый объект `Series` должен быть присвоен соответствующему аргументу `bottom`. Результатом будет сложенный столбчатый график.

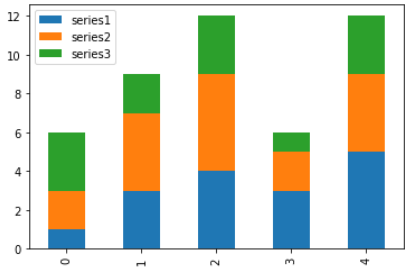
```
series1 =np.array([3,4,5,3])
series2 =np.array([1,2,2,5])
series3 =np.array([2,3,3,4])
index =np.arange(4)
plt.axis([-0.5,3.5,0,15])
plt.title('A Multiseries Stacked Bar Chart')
plt.bar(index,series1,color='r')
plt.bar(index,series2,color='b',bottom=series1)
plt.bar(index,series3,color='g',bottom=(series2+series1))
plt.xticks(index,['Jan18','Feb18','Mar18','Apr18'])
plt.show()
```



Сложенные столбчатые графики с Dataframe из padans

В случае со сложенными столбчатыми графиками очень легко представлять значения объектов Dataframe с помощью функции `plot()`. Нужно лишь добавить в качестве аргумента `stacked` со значением `True`.

```
importpandasas pd
data ={'series1':[1,3,4,3,5],
'series2':[2,4,5,2,4],
'series3':[3,2,3,1,3]}
df =pd.DataFrame(data)
df.plot(kind='bar',stacked=True)
plt.show()
```



При построении графиков можно задать стиль оформления графика:

```
matplotlib.style.use(style)
```

Список стилей можно получить с помощью команды:

```
print(matplotlib.style.available)
```

Как правило список стилей имеет вид:

```
'Solarize_Light2', '_classic_test_patch', 'bmh',
'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale',
'seaborn', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark',
'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep',
'seaborn-muted', 'seaborn-notebook', 'seaborn-paper',
'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks',
'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10'
```

По умолчанию используют стиль похожий на R

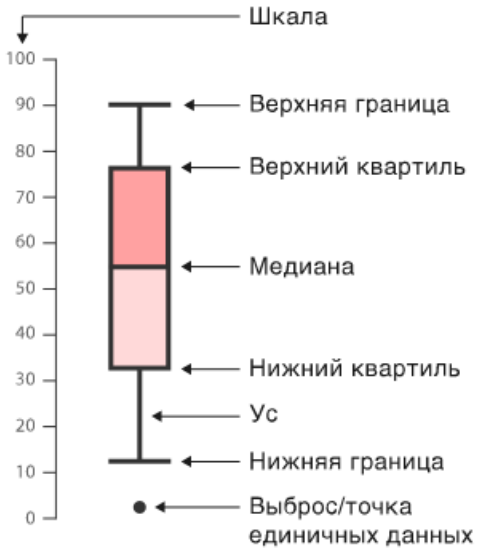
```
matplotlib.style.use('ggplot')
```

Альтернативой `matplotlib` является библиотека `seaborn`:

```
importseabornassns
sns.set(); # другой стиль оформления графика
```

Диаграмма размаха

ДР показывает данные о квантилях и выбросах.



Виды наблюдений, которые можно сделать на основе диаграммы размаха:

- каковы ключевые значения, например, средний показатель, медиана 25го перцентиля и так далее,
- существуют ли выбросы и каковы их значения,
- симметричны ли данные,
- насколько плотно сгруппированы данные,
- смещены ли данные и, если да, то в каком направлении.

Для визуализации данных в библиотеке Pandas можно использовать 3 способа:

- метод plot у DataFrame, принимающий в качестве аргумента kind, который определяет вид графика:

```
df.plot(kind='bar')
```

- функции для построения hist, bar, line (линейный) через метод plot:

```
data.plot.bar()
```

- напрямую обратиться к функциям bar, boxplot или hist:

```
data.bar()
```

Чтение и запись данных массивов в файлы

Важный аспект NumPy, которому пока не уделялось внимание — процесс чтения данных из файла. Это очень важный момент, особенно когда нужно работать с большим количеством данных в массивах. Это базовая операция анализа данных, поскольку размер набора данных почти всегда огромен, и в большинстве случаев не рекомендуется работать с ним вручную.NumPy предлагает набор функций, позволяющих специалисту сохранять результаты вычислений в текстовый или бинарный файл. Таким же образом можно считывать и конвертировать текстовые данные из файла в массив.

Загрузка и сохранение данных в бинарных файлах

NumPy предлагает пару функций, `save()` и `load()`, которые позволяют сохранять, а позже и получать данные, сохраненные в бинарном формате.

При наличии массива, который нужно сохранить, содержащего, например, результаты анализа данных, остается лишь вызвать функцию `call()` и определить аргументы: название файла и аргументы. Файл автоматически получит расширение `.npy`.

```
>>>data=([[0.86466285,0.76943895,0.22678279],
[0.12452825,0.54751384,0.06499123],
[0.06216566,0.85045125,0.92093862],
[0.58401239,0.93455057,0.28972379]])
>>>np.save('saved_data',data)
```

Когда нужно восстановить данные из файла `.npy`, используется функция `load()`. Она требует определить имя файла в качестве аргумента с расширением `.npy`.

```
>>>loaded_data=np.load('saved_data.npy')
>>>loaded_data
array([[0.86466285,0.76943895,0.22678279],
[0.12452825,0.54751384,0.06499123],
[0.06216566,0.85045125,0.92093862],
[0.58401239,0.93455057,0.28972379]])
```

Чтение файлов с табличными данными

Часто данные для чтения или сохранения представлены в текстовом формате (TXT или CSV). Их можно сохранить в такой формат вместо двоичного, потому что таким образом к ним можно будет получать доступ даже вне NumPy, с помощью других приложений. Возьмем в качестве примера набор данных в формате CSV (Comma-SeparatedValues — значения, разделенные запятыми). Данные здесь хранятся в табличной форме, а значения разделены запятыми.

pandas — библиотека, предназначенная для анализа данных, поэтому логично предположить, что она в первую очередь используется для вычислений и обработки данных. Процесс записи и чтения данных на/с внешние файлы — это часть обработки. Даже на этом этапе можно выполнять определенные операции, готовя данные к взаимодействию.

Первый шаг очень важен, поэтому для него представлен полноценный инструмент в библиотеке, называемый API I/O. Функции из него можно разделить на две категории: для чтения и для записи.

Чтение	**Запись**
read_csv	to_csv
read_excel	to_excel
read_hdf	to_hdf
read_sql	to_sql
read_json	to_json
read_html	to_html
read_stata	to_stata
read_clipboard	to_clipboard
read_pickle	to_pickle
read_msgpack	to_msgpack (экспериментальный)
read_gbq	to_gbq (экспериментальный)

CSV и текстовые файлы

Чаще всего текстовые файлы представлены в табличной форме. Если значения в колонке разделены запятыми, то это формат .csv (англ. commaseparatedvalues, значения, разделенные запятыми). Другие формы табличных данных могут использовать в качестве разделителей пробелы или табуляторы. Они хранятся в текстовых файлах разных типов (обычно с расширением .txt).

Такой тип файлов — самый распространенный источник данных, который легко расшифровывать и интерпретировать. Для этого pandas предлагает набор функций:

- `read_csv`
- `read_table`
- `to_csv`

Чтение данных из CSV или текстовых файлов

Самая распространенная операция по взаимодействию с данными при анализе данных — чтение их из файла CSV или как минимум текстового файла.

Для этого сперва нужно импортировать отдельные библиотеки.

```
import numpy as np
importpandasaspd
```

Чтобы сначала увидеть, как pandas работает с этими данными, используем файл CSV в рабочем каталоге `metal.csv`.

```
Date;Gold;Silver;Platinum;Palladium
18.09.2021 0:00:00;4120.07;53.68;2213.89;4770.7
17.09.2021 0:00:00;4148.59;54.66;2193.7;4732.06
16.09.2021 0:00:00;4219.32;55.83;2190;4609.54
15.09.2021 0:00:00;4181.7;55.17;2235.04;4811.41
14.09.2021 0:00:00;4200.92;55.41;2241.62;5023.67
11.09.2021 0:00:00;4210.48;56.2;2308.88;5183.86
10.09.2021 0:00:00;4221.14;56.77;2311.18;5301.84
```

Поскольку разделителем в файле выступают (;), можно использовать функцию `read_csv()` для чтения его содержимого и добавления в объект `Dataframe`.

```
values_df = pd.read_csv('metal.csv', delimiter=';')
```

```
print(values_df)
```

```
                   Date     Gold  Silver  Platinum  Palladium
0    18.09.2021 0:00:00  4120.07   53.68   2213.89    4770.70
1    17.09.2021 0:00:00  4148.59   54.66   2193.70    4732.06
2    16.09.2021 0:00:00  4219.32   55.83   2190.00    4609.54
3    15.09.2021 0:00:00  4181.70   55.17   2235.04    4811.41
4    14.09.2021 0:00:00  4200.92   55.41   2241.62    5023.67
..                   ...      ...     ...       ...        ...
907  16.01.2018 0:00:00  2433.40   31.02   1802.86    2043.84
908  13.01.2018 0:00:00  2425.60   30.95   1810.69    1976.29
909  12.01.2018 0:00:00  2418.56   31.40   1788.47    1982.71
910  11.01.2018 0:00:00  2416.66   31.19   1760.87    2009.54
911  10.01.2018 0:00:00  2411.72   31.49   1764.38    2022.99

[912 rowsx 5 columns]
```

Это простая операция. Файлы CSV — это табличные данные, где значения одной колонки разделены запятыми. Поскольку это все еще текстовые файлы, то подойдет и функция `read_table()`, но в таком случае нужно явно указывать разделитель.

```
values_tab = pd.read_table('metal.csv',sep=';')
print(values_tab)
```

```
                   Date     Gold  Silver  Platinum  Palladium
0    18.09.2021 0:00:00  4120.07   53.68   2213.89    4770.70
1    17.09.2021 0:00:00  4148.59   54.66   2193.70    4732.06
2    16.09.2021 0:00:00  4219.32   55.83   2190.00    4609.54
3    15.09.2021 0:00:00  4181.70   55.17   2235.04    4811.41
4    14.09.2021 0:00:00  4200.92   55.41   2241.62    5023.67
..                   ...      ...     ...       ...        ...
907  16.01.2018 0:00:00  2433.40   31.02   1802.86    2043.84
908  13.01.2018 0:00:00  2425.60   30.95   1810.69    1976.29
909  12.01.2018 0:00:00  2418.56   31.40   1788.47    1982.71
910  11.01.2018 0:00:00  2416.66   31.19   1760.87    2009.54
911  10.01.2018 0:00:00  2411.72   31.49   1764.38    2022.99

[912 rowsx 5 columns]
```

В этом примере все заголовки, обозначающие названия колонок, определены в первой строчке.

Также можно самостоятельно определить колонки, присвоив список меток параметру `names`.

```
values_df = pd.read_csv('metal.csv', delimiter=';', names=['Date', 'Silver'])
print(values_df
```

```
                                    Date     Silver
Date                Gold    Silver  Platinum  Palladium
18.09.2021 0:00:00 4120.07 53.68    2213.89     4770.7
17.09.2021 0:00:00 4148.59 54.66     2193.7    4732.06
16.09.2021 0:00:00 4219.32 55.83       2190    4609.54
15.09.2021 0:00:00 4181.7  55.17    2235.04    4811.41
...                                      ...        ...
16.01.2018 0:00:00 2433.4  31.02    1802.86    2043.84
13.01.2018 0:00:00 2425.6  30.95    1810.69    1976.29
12.01.2018 0:00:00 2418.56 31.4     1788.47    1982.71
11.01.2018 0:00:00 2416.66 31.19    1760.87    2009.54
10.01.2018 0:00:00 2411.72 31.49    1764.38    2022.99

[913 rowsx 2 columns]
```

Чтение и запись в файлы MicrosoftExcel

Данные очень легко читать из файлов CSV, но они часто хранятся в табличной форме в формате Excel.pandas предоставляет специальные функции для работы с ним:

- `to_excel()`
- `read_excel()`

Функция `read_excel()` может читать из файлов Excel 2003 (.xls) и Excel 2007 (.xlsx). Дляработы с Excelнеобходимо установить пакеты `xlrd` и`openpyxl`:

```
>>pipinstallxlrdopenpyxl
```

Для начала откроем файл Excel и введем данные со следующий таблиц. Разместим их в листах `sheet1` и `sheet2`. Сохраним файл как `excel_data.xlsx`.

	white	red	green	black
a	12	23	17	18
b	22	16	19	18
c	14	23	22	21

	yellow	purple	blue	orange
A	11	16	44	22
B	20	22	23	44
C	30	31	37	32

Для чтения данных из файла XLS нужно всего лишь конвертировать его в `Dataframe`, используя для этого функцию `read_excel()`.

```
pd.read_excel('excel_data.xlsx')
```

По умолчанию готовый объект pandas`Dataframe` будет состоять из данных первого листа файла. Но если нужно загрузить и второй, то достаточно просто указать его номер (индекс) или название в качестве второго аргумента.

```
pd.read_excel('excel_data.xlsx','Sheet2')
```

	yellow	purple	blue	orange
A	11	16	44	22
B	20	22	23	44
C	30	31	37	32

```
>>>pd.read_excel('excel_data.xlsx',1)
```

	yellow	purple	blue	orange
A	11	16	44	22
B	20	22	23	44
C	30	31	37	32

Запись работает по тому же принципу. Для конвертации объекта `Dataframe` в Excel нужно написать следующее.

```
frame = pd.DataFrame(np.random.random((4,4)), index =
['exp1','exp2','exp3','exp4'],columns =
['Jan2015','Fab2015','Mar2015','Apr2005'])

frame.to_excel('data2.xlsx')
```

	Jan2015	Feb2015	Mar2015	Apr2015
exp1	0.671044	0.437715	0.497103	0.070595
exp2	0.864018	0.575196	0.240343	0.471081
exp3	0.957986	0.311648	0.381975	0.622556
exp4	0.407909	0.015926	0.180611	0.579783

В рабочей директории будет создан файл с соответствующими данными.

Предварительная работа.Установка библиотеки инструментов

NumPy

NumPy – это один из основных пакетов для научных вычислений в Python. Он содержит функциональные возможности для работы с многомерными массивами, высокоуровневыми математическими функциями (операции линейной алгебры, преобразование Фурье, генератор псевдослучайных чисел и др.).

pandas

pandas – библиотека Python для обработки и анализа данных. Она построена на основе структуры данных, называемой DataFrame. Структура DataFrame представляет собой таблицу, похожую на электронную таблицу Excel. Библиотека pandas предлагает большой спектр методов по работе с этой таблицей. В отличие от NumPy, который требует, чтобы все записи в массиве были одного и того же типа, в pandas каждый столбец может иметь отдельный тип. Еще одним преимуществом библиотеки pandas является ее способность работать с различными форматами файлов и баз данных, например, с файлами SQL, Excel и CSV.

SciPy

SciPy – это набор функций для научных вычислений в Python. Он предлагает процедуры линейной алгебры, оптимизацию функций, обработку сигналов, специальные математические функции и статистические функции. scikit-learn использует набор функций SciPy для реализации своих алгоритмов.

matplotlib

matplotlib – это основная библиотека для построения научных графиков в Python. Она включает функции для создания высококачественных визуализаций типа линейных диаграмм, гистограмм, диаграмм разброса и т.д.

Scikit-learn

scikit-learn – проект с открытым исходным кодом, базируется на двух библиотеках для научных вычислений NumPy и SciPy. Он содержит ряд современных алгоритмов машинного обучения, а также полную документацию по каждому алгоритму.

В scikit-learn массив NumPy – это основная структура данных. scikit-learn принимает данные в виде массивов NumPy. Любые данные, которые вы используете, должны быть преобразованы в массив NumPy.Для понимания принципов работы и использования scikit-learn важно рассмотреть базовые библиотеки для научных вычислений NumPy, SciPy, pandas и графическую библиотеку matplotlib.

Установка пакетов

Если у вас уже установлен Python, вы можете использовать `pip`или `pip3` для установки всех этих пакетов (требуется подключение к сети Internet):

```
>> pip install numpypandas matplotlib scipyipythonjupiterscikit-learn
seaborn
```

Или по-отдельности:

```
>>pipinstallnumpy
>> pip install scipy
>> pip install matplotlib
```

…

При необходимости, в дальнейшем, можно оперативно добавлять дополнительные библиотеки.

Практическая работа № 1. Наборы данных. Извлечение и визуализация данных.

Тема работы: изучение методов извлечения данных из наборов с использованием средств библиотеки `pandas Python`.

Цель работы: Научиться использовать на практике методы программирования в программной среде `Python` на примере использования методов извлечения данных, сведений о данных и визуализации данных.

Методические указания по выполнению лабораторной работы

Pandas — библиотека для обработки и анализа данных, которая предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временными рядами (уровень абстракции данных для объединения и преобразования данных). Работа Pandas с данными строится поверх библиотеки NumPy.

Источниками данных для структур библиотеки обычно служат или файлы с данными или базы данных. Наиболее распространённые форматы для хранения данных – табличные файлы csv и Excel.

Для работы с данными в Pandas используются 2 структуры:

- Series (серии),
- DataFrame (фреймы данных).

Структура или объект Series - одномерный массив или список с ассоциированными метками (индексами), т.е. этот объект подобен ассоциативному массиву или словарю в Python.

Конструктор класса Series выглядит следующим образом:

```
pandas.Series(data=None, index=None, dtype=None, name=None, copy=False,
fastpath=False)
```

- data – массив, словарь или скалярное значение, на базе которого будет построен Series;
- index – список меток, который будет использоваться для доступа к элементам Series. Длина списка должна быть равна длине data;
- dtype – объект numpy.dtype, определяющий тип данных;
- copy – создает копию массива данных, если параметр равен True в ином случае ничего не делает.

Создать структуру Series можно на базе различных типов данных:

- словари Python;
- списки Python;
- массивы из numpy: ndarray;
- скалярные величины.

Объект DataFrame является табличной структурой данных, в нем есть строки и столбцы. Столбцами в объекте DataFrame выступают объекты Series, строки которых являются их непосредственными элементами.

Объект DataFrame имеет индексы по строкам и по столбцам. Если индекс по строкам явно не задан (например, колонка по которой нужно их строить), то Pandas задаёт целочисленный индекс RangeIndex от 0 до N-1, где N это количество строк в таблице.

Конструктор класса DataFrame выглядит так:

```
class pandas.DataFrame(data=None, index=None, columns=None, dtype=None,
copy=False)
```

- data – массив ndarray, словарь (dict) или другой DataFrame;
- index – список меток для записей (имена строк таблицы);
- columns – список меток для полей (имена столбцов таблицы);
- dtype – объект numpy.dtype, определяющий тип данных;
- copy – создает копию массива данных, если параметр равен True в ином случае ничего не делает.

Структуру DataFrame можно создать на базе:

- словаря (dict) в качестве элементов которого должны выступать: одномерные ndarray, списки, другие словари, структуры Series;
- двумерные ndarray;
- структуры Series;
- структурированные ndarray;
- другие DataFrame.

Загрузка данных из табличного файла

CSV (от англ. Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделённых запятыми.

С файлом можно работать через текстовый или табличный редактор (например MS Excel).

Для загрузки данных их табличных файлов можно использовать объект DataFrame. Для этого, в первую очередь, подключаем библиотеку:

```
import pandas as pd
```

Затем используем метод read_csv():

```
fix_df = pd.read_csv('c://bikes.csv',
sep=';',
encoding='latin1')
```

В методе требуется указать первым параметром путь до имени файла, далее по необходимости параметры sep - задаёт символ-разделитель полей в файле (по умолчанию разделитель запятая), names - список названий колонок, если он не задан в файле, index_col -- номер колонки с индексом, decimal - символ-разделитель для знаков после запятой, encoding – кодировку файла и т.д.

Каждая строчка набора данных является одним наблюдением или одним объектом в задачи и их требуется различать. Для этого используется индекс, по умолчанию, если индекс не указан каждая строка нумеруется. В качестве индекса может быть использован один из столбцов таблицы:

```
fix_df1 = pd.read_csv('d://bikes.csv',
sep=';',
encoding='latin1',
index_col='Date' )
```

Можно указать какой тип данных должен быть у столбца:

```
fix_df2 = pd.read_csv('d://bikes.csv',
sep=';', encoding='latin1',
dtype={'Date':str} )
```

Также можно указать какой столбец должен считываться как дата:

```
fix_df3 = pd.read_csv('d://bikes.csv',
sep=';', encoding='latin1',
parse_dates=['Date'],
dayfirst=True,
index_col='Date')
```

Экспорт (выгрузка) данных

Данные датафрейма могут быть изменены в процессе обработки и их потребуется сохранить в табличный файл. Для этого можно использовать команду:

```
dataframe.to_csv('file_name.csv')
```

Метод `.to_csv` имеет ряд входных параметров, которые могут указывать формат выходного файла csv:

- кодировка, например encoding='utf-8',
- разделитель значений sep, по умолчанию ‘,’,
- запись имен строк (индексы) по умолчанию True,

- и другие.

Вывод информации о наборе данных

Операция	Метод
Для быстрого просмотра первых 5 строк набора данных можно использовать метод head():	`fix_df.head()`
Для получения последних 5 строк – метод tail():	`fix_df.tail()`
Для получения определённого количества случайных строк из набора данных используют метод sample(количество):	`fix_df.sample(2)`
Для вывода определённого количества строк можно указать число:	`fix_df[:3]`
Для ввода данных из определённого столбца можно указать его название и количество выводимых значений:	`fix_df['Berri 1'][:10]`
Размерность фрейма данных можно получить, используя свойство фрейма данных shape:	`fix_df.shape`
Количество строк можно найти, применив функцию len() длина, к индексу фрейма данных или к фрейму:	`len(fix_df.index)` `len(fix_df)`
Для получения информации о типах столбцов фрейма данных используется dtypes:	`fix_df.dtypes`
Для получения информации о статистических оценках значений фрейма используют метод describe():	`fix_df.describe(include='all')`

Обработка пустых значений

Пустое значение для обработки данных является особым случаем, требующем обработки: исключения или заполнения. Если пропущенных значений не очень много, можно их заполнить. Если пропущенных значений в каком-то столбце много, то следует исключить столбец из рассмотрения, так как данных по нему недостаточно.

Для работы с пропечёнными значениями в библиотеке есть методы:

- isnull() — генерирует булеву маску для отсутствующих значений,
- dropna() — фильтрация данных по отсутствующим значениям,
- fillna() — замена пропусков, аргументы method=’ffill’ и method=’bfill’ определяют какими значениями будут заполняться пропуски (предыдущими или последующими в массиве).

Методы доступны как для объектов Series так и для dataFrame (с выбором столбца).

Для получения количества пустых значений в столбцах можно использовать метод isnull():

```
# Возвращает количество пропущенных значений, содержащихся в каждом
столбце
df.isnull ()df.isnull (). sum ()
```

Метод fillna() не изменяет текущую структуру, он возвращает структуру DataFrame, созданную на базе существующей, с заменой NaN значений на те, что переданы в метод в качестве аргумента. Операция по умолчанию df.fillna () (inplace = False) не является внутренней, то есть она не изменяет напрямую исходный фрейм данных, а создает копию и изменяет копию. В качестве значения для заполнения могут использоваться:

- ближайшее ненулевое значение,
- скользящее среднее,
- следующе или предыдущее значение и др.

Для удаления объектов, которые содержат значения *NaN,* используется метод *dropna()*, в зависимости от параметров метода можно удалять весть столбец с пстыми значениями, только строки с пустыми значениями, указывать ограничение на возможное число пустых значений в строке.

```
# Удалить строки с пропущенными значениями напрямую
df.dropna ()
```

```
# Прямое удаление столбцов с пропущенными значениями
df.dropna (axis = 1)
# Удалять только строки с пропущенными значениями
df.dropna (how = 'all')
# Сохранять строки с как минимум 4 пропущенными значениями
df.dropna (thresh = 4)
```

Работа с индексами набора данных

При больших объемах данных сортировка и поиск требуют достаточного времени. Для уменьшения времени для выполнения этих операций используется механизм индексов, аналогично индексам в базах данных.

Индекс можно указать сразу, при создании набора данных с помощью параметра index_col:

```
fix_df1 = pd.read_csv('d://bikes.csv',
sep=';',
encoding='latin1',
index_col='Date' )
```

Или индекс можно установить позже с помощью метода set_index():

```
df_with_index = df.set_index(['key'])
index_moved_to_col.set_index('Sector').head()
```

В случае, если индекс устанавливается для нескольких полей, он имеет структуру – иерархию (слои), и имеет значение порядок следования полей. Можно просмотреть количество уровней в индексе и изменить порядок слоев.

Пример:

```
# индексируем датафрейм data по столбцам Sector и Symbol
multi_fi = reindexed.set_index(['Sector', 'Symbol'])
len(multi_fi.index.levels)
# изменение порядка уровней индекса
multi_fi.reorder_levels([1, 0], axis=0).head()
```

Индекс может быть сброшен с помощью метода reset_index():

```
index_moved_to_col = sp500.reset_index()
```

Задание:

1. Выберите источник данных, предварительно рассмотрите ресурсы с открытыми наборами данных (Приложение А, или выбранный самостоятельно источник данных).

2. Для выбранного набора данных реализовать:

- чтение набора в таблицу записями типа DataFrame;
- получение информации о наборе данных, полях, типах данных, статистиках;
- вывести данные в консоль;
- преобразовать записи в массив;
- выбрать способ визуализации данных и представить их графически.

Пример выполнения задания:

```
# Этап 1. Получение данных
# Изучим данные, предоставленные сервисом
# Импорт библиотек
# <импорт библиотеки pandas>
importpandasaspd

# <Чтение данных из файла в таблицу записями типа DataFrame>
# Вариант 1
values_df = pd.read_csv('metal.csv', delimiter=';')
# Вариант 2
#values_df = pd.read_table('metal.csv', sep=';')
```

```
# Этап 2. Извлечение сведений о данных
# <получение общей информации о данных в таблице>
print('Общая информаця о данных в таблице')
print(values_df.info())
# <вывод данных таблицы>
print('Вывод данных таблицы')
print(values_df)
# <перечень названий столбцов таблицы values_df способ 1>
print('Вывод названий столбцов таблицы. способ 1')
print(values_df.columns)
# <перечень названий столбцов таблицы values df способ 2>
print('Вывод названий столбцов таблицы. способ 2')
print(list(values_df))
# Проверим данные на наличие пропусков
# <суммарное количество пропусков, выявленных методом isnull() в таблице
values df>
print('Количество пустых строк')
print(values_df.isnull().sum())
# <получение суммарного количества дубликатов в таблице values_df>
# для временного ряда - не совсем корректная информация
print('Количество строк-дубликатов')
print(values_df.duplicated().sum())
# Метод describe() позволяет собрать некоторую статистику по каждому
числовому признаку
print('Статистика по признакам')
print(values_df.describe())
# <получение первых 10 строк таблицы values_df>
print('1-е 10 значений таблицы')
print(values_df.head(10))

# Этап 3. Преобразование данных в массив
# <импорт библиотеки numpy>
importnumpyasnp
# <приведение типа DataFrame к типу ndArray и выбор 2-го столбца данных
('Silver')>
values_arr2 = np.array(values_df)[:, 2]
# <вывод данных массива>
print('Вывод данных массива')
print(values_arr2)
# <приведение типа DataFrame к типу ndArray и выбор 2-го и 3-го столбцов
данных ('Silver', 'Platinum')>
values_arr23 = np.array(values_df)[:, 2:4]
# <вывод данных массива>
print('Вывод данных массива')
print(values_arr23)

# Этап 4. Визуализация данных массива
# Построение графиков
# <импорт библиотеки matplotlib>
importmatplotlib
importmatplotlib.pyplotaspltfrompylabimportplot

# <задание типа графика>
matplotlib.style.use('ggplot')
importseabornassns
#sns.set(); # другой вид графиков по умолчанию

plt.figure()
x = np.linspace(values_arr2[0], values_arr2[values_arr2.size-1] ,
values_arr2.size)
plot(x, values_arr2,'g')
plt.show()
```

Результат выполнения:

```
C:\Python\Projects\venv\Scripts\python.exe "C:/Python/Projects/Practic 1.1.py"
Общая информаця о данных в таблице
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 912 entries, 0 to 911
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       912 non-null    object
 1   Gold       912 non-null    float64
 2   Silver     912 non-null    float64
 3   Platinum   912 non-null    float64
 4   Palladium  912 non-null    float64
dtypes: float64(4), object(1)
memory usage: 32.1+ KB
None
Вывод данных таблицы
                   Date     Gold  Silver  Platinum  Palladium
0    18.09.2021 0:00:00  4120.07   53.68   2213.89    4770.70
1    17.09.2021 0:00:00  4148.59   54.66   2193.70    4732.06
2    16.09.2021 0:00:00  4219.32   55.83   2190.00    4609.54
3    15.09.2021 0:00:00  4181.70   55.17   2235.04    4811.41
4    14.09.2021 0:00:00  4200.92   55.41   2241.62    5023.67
..                   ...      ...     ...       ...        ...
907  16.01.2018 0:00:00  2433.40   31.02   1802.86    2043.84
908  13.01.2018 0:00:00  2425.60   30.95   1810.69    1976.29
909  12.01.2018 0:00:00  2418.56   31.40   1788.47    1982.71
910  11.01.2018 0:00:00  2416.66   31.19   1760.87    2009.54
911  10.01.2018 0:00:00  2411.72   31.49   1764.38    2022.99

[912 rows x 5 columns]
Вывод названий столбцов таблицы. способ 1
Index(['Date', 'Gold', 'Silver', 'Platinum', 'Palladium'], dtype='object')
Вывод названий столбцов таблицы. способ 2
['Date', 'Gold', 'Silver', 'Platinum', 'Palladium']
Количество пустых строк
Date         0
Gold         0
Silver       0
Platinum     0
Palladium    0
dtype: int64
Количество строк-дубликатов
0
Статистика по признакам
GoldSilverPlatinumPalladium
count   912.000000  912.000000   912.000000   912.000000
mean   3408.429342   42.478015  2029.763575  3979.578542
std     806.406054   13.233652   377.442826  1607.828190
min    2379.100000   29.690000  1522.760000  1684.600000
25%    2653.975000   31.660000  1763.462500  2603.762500
50%    3091.840000   35.755000  1846.370000  3632.340000
75%    4243.552500   58.620000  2239.035000  5566.380000
max    4887.700000   72.200000  3095.570000  7227.220000
1-е 10 значенийтаблицы
                 Date     Gold  Silver  Platinum  Palladium
0  18.09.2021 0:00:00  4120.07   53.68   2213.89    4770.70
1  17.09.2021 0:00:00  4148.59   54.66   2193.70    4732.06
2  16.09.2021 0:00:00  4219.32   55.83   2190.00    4609.54
3  15.09.2021 0:00:00  4181.70   55.17   2235.04    4811.41
4  14.09.2021 0:00:00  4200.92   55.41   2241.62    5023.67
5  11.09.2021 0:00:00  4210.48   56.20   2308.88    5183.86
6  10.09.2021 0:00:00  4221.14   56.77   2311.18    5301.84
7  09.09.2021 0:00:00  4245.35   57.41   2363.58    5565.39
8  08.09.2021 0:00:00  4260.95   57.05   2386.08    5638.12
9  07.09.2021 0:00:00  4277.87   57.95   2413.54    5690.23
Вывод данных массива
[53.68 54.66 55.83 55.17 55.41 56.2 56.77 57.41 57.05 57.95 56.34 56.6

… … … …

30.56 29.98 29.97 30.23 30.45 30.59 30.44 30.33 29.69 30.04 30.21 30.48
 30.54 30.44 30.22 30.12 30.0 30.35 29.99 30.19 29.72 29.83 30.11 29.84
 29.86 29.99 30.18 30.38 30.5 30.48 30.17 30.75 30.51 30.51 30.57 30.95
 30.77 31.04 31.2 30.97 31.17 31.24 31.37 31.49 31.45 30.91 30.78 30.9
 31.02 31.09 31.41 31.1 31.41 31.02 30.95 31.4 31.19 31.49]
```

```
Выводданныхмассива
[[53.68 2213.89]
 [54.66 2193.7]
 [55.83 2190.0]
 ...
 [31.4 1788.47]
 [31.19 1760.87]
 [31.49 1764.38]]

Process finished with exit code 0
```

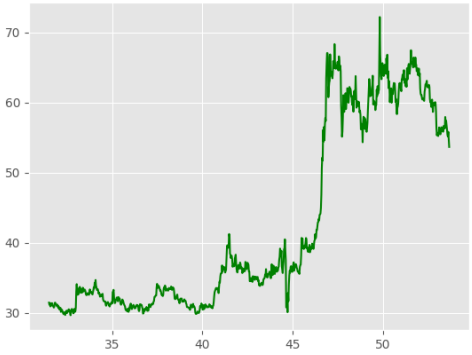


Рис. Пример визуализации

Практическая работа № 2. Сглаживание и прогнозирование временных рядов

Тема работы: изучение методов программирования в программной среде Pythonна примере сглаживания эмпирических данных.

Цель работы: Научиться использовать на практике методы программирования в программной средеPythonна примере использования простейших алгоритмов линейного и нелинейного сглаживания данных и их численного дифференцирования.

Множество значений статистического показателя в последовательные моменты времени называют временным (динамическим) рядом.

Основная задача сглаживания временного ряда состоит в устранении неверных и сезонных колебаний для определения асимптотического поведения временного ряда – тренда. Сглаженные данные могут использоваться для прогноза и как часть анализа сезонных колебаний.

Задание: имеется ряд экспериментально измеренных значений y_i функции $u = u(x)$ в равноотстоящих точках x_i. Результаты измерений содержат «экспериментальный шум» - случайные ошибки ε существенной величины, причем уровень шума не зависит от x, так что $y_i = u(x_i) + \varepsilon$.

Требуется сгладить результаты измерений, используя алгоритмы линейного и нелинейного сглаживания. Выполнить численное дифференцирование исходных экспериментальных $y(x)$ и сглаженных $\tilde{y}(x)$ данных, используя операторы второго и четвертого порядка точности. Сравнить результаты сглаживания с истинной функцией $u(x)$. Определить оптимальный шаг численного дифференцирования и вычислить первую производную сглаженной и неслаженной функций с данным шагом.

Теоретическая часть

Некоторые задачи, возникающие при анализе и интерпретации опытных данных, не требуют построения единой аналитической формулы во всем диапазоне изменения переменной x. Например, для численного дифференцирования важно лишь устранить «шум» эксперимента, сохранив информацию об истинной функции. Для этой цели применяется сглаживание эмпирических данных, т. е. замена исходной таблицы опытных точек другой таблицей близких к ним точек, лежащих на достаточно гладкой кривой.

При сглаживании используется метод наименьших квадратов и аппроксимирующие многочлены различных степеней. Если используется многочлен первой степени, сглаживание называется *линейным*, в противном случае – *нелинейным*. Количество точек для сглаживания берут нечетным, а группы точек – «скользящими» вдоль всей таблицы.

Линейное сглаживание по трем точкам приводит к формулам:

$$\tilde{y}_i = \frac{1}{3}(y_{i-1} + y_i + y_{i+1}),\ i = 1, 2, 3, \dots, n-1,$$

соотношения для сглаживания первой и последней точек:

$$\tilde{y}_0 = \frac{1}{6}(5y_0 + 2\tilde{y}_1 - \tilde{y}_2),$$

$$\tilde{y}_n = \frac{1}{6}(5y_n + 2\tilde{y}_{n-1} - \tilde{y}_{n-2}).$$

Формулы линейного сглаживания по пяти точкам имеют вид:

$$\tilde{y}_i = \frac{1}{5}(y_{i-2} + y_{i-1} + y_i + y_{i+1} + y_{i+2}),\ \ i = 2,3,4, \dots, n-2,$$

соотношения для сглаживания первых двух и последних двух точек:

$$\tilde{y}_1 = \frac{1}{10}(4y_0 + 3y_1 + 2\tilde{y}_2 + \tilde{y}_3),$$

$$\tilde{y}_0 = \frac{1}{5}(3y_0 + 2\tilde{y}_1 + \tilde{y}_2 - y_3),$$

$$\tilde{y}_{n-1} = \frac{1}{10}(4y_n + 3y_{n-1} + 2\tilde{y}_{n-2} + \tilde{y}_{n-3}),$$

$$\tilde{y}_n = \frac{1}{5}(3y_n + 2\tilde{y}_{n-1} + \tilde{y}_{n-2} - \tilde{y}_{n-4}).$$

Формулы нелинейного сглаживания (многочленом третьей степени) по семи точкам:

$$\tilde{y}_i = \frac{1}{21}(-2y_{i-3} + 3y_{i-2} + 6y_{i-1} + 7y_i + 6y_{i+1} + 3y_{i+2} - 2y_{i-3}),\ i = 3,4,5,\dots,n-3,$$

соотношения для сглаживания первых трех и последних трех точек:

$$\tilde{y}_2 = \frac{1}{42}(-4y_0 + 16y_1 + 19y_2 + 12\tilde{y}_3 + 2\tilde{y}_4 - 4\tilde{y}_5 + \tilde{y}_6),$$

$$\tilde{y}_1 = \frac{1}{42}(8y_0 + 19y_1 + 16\tilde{y}_2 + 6\tilde{y}_3 - 4\tilde{y}_4 - 7\tilde{y}_5 + 4\tilde{y}_6),$$

$$\tilde{y}_0 = \frac{1}{42}(39y_0 + 8\tilde{y}_1 - 4\tilde{y}_2 - 4\tilde{y}_3 + \tilde{y}_4 + 4\tilde{y}_5 - 2\tilde{y}_6),$$

$$\tilde{y}_{n-2} = \frac{1}{42}(-4y_n + 16y_{n-1} + 19y_{n-2} + 12\tilde{y}_{n-3} + 2\tilde{y}_{n-4} - 4\tilde{y}_{n-5} + \tilde{y}_{n-6})$$

$$\tilde{y}_{n-1} = \frac{1}{42}(8y_n + 19y_{n-1} + 16\tilde{y}_{n-2} + 6\tilde{y}_{n-3} - 4\tilde{y}_{n-4} - 7\tilde{y}_{n-5} + 4\tilde{y}_{n-6})\ \tilde{y}_n = \frac{1}{42}(39y_n + 8\tilde{y}_{n-1} - 4\tilde{y}_{n-2} -$$

$$4\tilde{y}_{n-3} + \tilde{y}_{n-4} + 4\tilde{y}_{n-5} - 2\tilde{y}_{n-6}).$$

Задача численного дифференцирования

Применительно к экспериментальным данным задача численного дифференцирования наиболее просто решается в том случае, когда значения функции $y = y(x)$ измерены в равноотстоящих точках:

$$y_i = y(x_i),\ x_i = x_1 + i \cdot h,\ i = 1,2,\dots,n.$$

Требуется вычислить значения производной y_i' в тех же точках. Другими словами, по таблице функции с постоянным шагом h требуется составить таблицу ее производных с тем же шагом.

На результаты численного дифференцирования очень большое влияние оказывает «шум» эксперимента. Относительно небольшие ошибки в измерении исходной функции $y = y(x)$ могут привести к большим ошибкам в ее производной. Причина подобного искажения результатов связана отнюдь не с несовершенством методов вычисления производных, а с тем обстоятельством, что сама операция *численного дифференцирования приближенно заданной функции* является *некорректной*.

Поэтому при наличии в значения функции случайных ошибок необходимо предварительно сгладить исходные данные, а затем уже применять те или иные методы численного дифференцирования.

Чаще других при численном дифференцировании используется формула центральной разностной производной.

$$y_i' = \frac{y_{i+1} - y_{i-1}}{2h}.$$

Она применяется при $i = 1,2,3,\dots,n-1$, а в начальной и конечной точке применяются формулы односторонних производных

$$y_0' = \frac{-3y_0 + 4y_1 - y_2}{2h}, \quad y_n' = \frac{3y_n - 4y_{n-1} + y_{n-2}}{2h}.$$

Эти формулы имеют *второй порядок точности* по h.

Результаты численного дифференцирования сильно «зашумленных» данных существенно зависят от качества сглаживающего алгоритма.

Прогнозирование временного ряда. Модели прогнозирования временного ряда

МоделиARIMA

В этом разделе мы разработаем модели интегрированной скользящей средней с авторегрессией или ARIMA для решения этой проблемы.

Мы подойдем к этому в четыре этапа:

1. Разработка вручную настроенной модели ARIMA.
2. Использование сетки поиска ARIMA, чтобы найти оптимизированную модель.
3. Анализ остаточных ошибок прогноза для оценки любого отклонения в модели.
4. Исследуйте улучшения в модели с помощью силовых преобразований.

Сконфигурированная вручную ARIMA

Несезонный ARIMA (p, d, q) описывает интегрированные процессы авторегрессии и скользящей средней. Модель требует трех параметров и традиционно настраивается вручную, где d – порядок разности ($\Delta^d Y_t = (Y_t — Y_{t—1})^d$), p – порядок авторегрессии, q – порядок скользящего среднего.

Анализ данных временного ряда предполагает, что мы работаем со стационарным временным рядом.

Мы оценим эффективность прогнозов, используя среднеквадратичную ошибку (СКО, англ. root-mean-squareerror, RMSE).

Мы можем вычислить RMSE, используя вспомогательную функцию `mean_squared_error()` из библиотеки `scikit-learn`, которая вычисляет среднеквадратичную ошибку между списком ожидаемых значений (набор тестов) и списком прогнозов. Затем мы можем взять квадратный корень из этого значения, чтобы дать нам СКО.

Например:

```
from sklearn.metrics import mean_squared_error
from math import sqrt
...
test = ...
predictions = ...
mse = mean_squared_error(test, predictions)
rmse = sqrt(mse)
print(СКО: %.3f' % rmse)
```

Тестовая стратегия

Модели-кандидаты будут оцениваться с использованием предварительной проверки.Предварительная проверка будет работать следующим образом:

- Первые 50% набора данных будут отложены для обучения модели.
- Оставшиеся 50% набора данных будут повторяться и тестировать модель.
- Для каждого шага в тестовом наборе данных:
  - модель будет обучена.
  - сделан одношаговый прогноз, и прогноз сохранен для последующей оценки.
  - фактическое наблюдение из набора тестовых данных будет добавлено в набор обучающих данных для следующей итерации.
- Прогнозы, сделанные во время итерации тестового набора данных, будут оценены и получена оценка СКО.

Учитывая небольшой размер данных, мы позволим модели пройти переподготовку с учетом всех доступных данных перед каждым прогнозом.

Во-первых, мы можем напрямую разбить набор данных на обучающие наборы `train` и тесты `test`. Преобразуем загруженный набор данных в float32, если в загруженных данных есть типы данных String или Integer.

```
# подготовка данных
X = series.values
X = X.astype('float32')
train_size = int(len(X) * 0.50)
train, test = X[0:train_size], X[train_size:]
```

Далее выполним итерации по временным шагам в тестовом наборе данных. Набор данных train хранится в списке Python, так как нам нужно легко добавлять новое наблюдение к каждой итерации, и конкатенация массивов Numpy выглядит как перебор.

Прогноз, сделанный моделью, называется `yhat`.

```
# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
  # predict
  yhat = ...
  predictions.append(yhat)
  # наблюдение
  obs = test[i]
  history.append(obs)
  print('>Прогноз = %.3f, Ожидаемоезначение = %3.f' % (yhat, obs))
```

Порядок выполнения задания

1. Из файла данных `metal.csv` извлеките данные используя функции чтения данных (разделитель данных в строке – (;), шаг 1 единица (день)). Рабочим массивом примите данные 2-го столбца.

2. Напишите функцию для алгоритма линейного сглаживания данных по трем точкам; изобразите на одном графике исходные (y), сглаженные и не зашумлённые данные.

3. Напишите функцию для алгоритма линейного сглаживания данных по пяти точкам, постройте графики исходных, сглаженных и не зашумлённых данных.

4. Напишите функцию для алгоритма нелинейного сглаживания по семи точкам; результаты сглаживания изобразите графически, аналогично п.3, п.4.

5. Сравните (графически) качество сглаживания с использованием линейного и нелинейного алгоритма.

6. Поскольку на практике часто используется повторное сглаживание, выполните двукратное и трехкратное сглаживание и прокомментируйте полученные результаты.

7. Используя формулы второго порядка точности, выполните численное дифференцирование исходных, сглаженных и не зашумлённых данных. Сравните полученные результаты.

8. Сделайте выводы по проделанной работе.

9[1]. Исследуйте прогнозные моделиARIMA.

Пример листинга выполнения п.1., п.2.

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import datetime

matplotlib.style.use('ggplot')

# функциясглаживания (фильтрации) по 3-мточкам
def three_point(points):
    points1 = np.zeros(len(points))
    size = len(points)
for i in range(0,len(points)-1):
        points1[i] = (points[i-1]+points[i]+points[i+1])/3
points1[0] = (5*points[0] + 2*points1[1] - points1[2])/6
points1[size-1] = (5*points[size-1] + 2*points1[size-2] - points1[size-
3])/6
```

[1] Дополнительное задание (самостоятельная работа)

```
return points1

# функциявычислениясреднеквадратичнойошибки
def sqerr(p1, p2):
    err1 = p1-p2
    err1 = err1**2
err2 = sum(err1)
return err2/len(p1)

# текстпрограммы
values_df = pd.read_csv('metal.csv', delimiter=';')
print(values_df)
index = 2
values_arr = np.array(values_df)[:,index]
X = values_arr # данныедляпрогнозноймодели
print(values_arr)
points3 = three_point(values_arr)
err3 = sqerr(values_arr, points3)
print(err3)

# <поосиабсцисс - дата>
values_df.Date = values_df.Date.apply(lambda x:
datetime.datetime.strptime(x, '%d.%m.%Y %H:%M'))
plt.figure()
values_df.index = values_df.Date
values_arr = pd.DataFrame(np.array(values_df)[:,index])
values_arr.index = values_df.Date
plt.plot(values_arr,'b', label = 'Исходный')
points3_df = pd.DataFrame(points3)
points3_df.index = values_df.Date
plt.plot(points3_df, 'r', label = 'Сглаженый')
plt.legend()

# Прогнозпомодели ARIMA
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt
# подготовка данных
X = X.astype('float32')
train_size = int(len(X) * 0.5)
train, test = X[0:train_size], X[train_size:len(X)]
history = [x for x in train]
predictions = list()
for i in range(len(test)):
     model = ARIMA(history, order=(5, 1, 0))
     model_fit = model.fit(disp=0)
     yhat = model_fit.forecast()[0]
     predictions.append(yhat)
# наблюдение
obs = test[i]
     history.append(obs)
#print('>Прогноз = %.3f, Ожидаемое значение = %3.f' % (yhat, obs))

rmse = sqrt(mean_squared_error(test, predictions))
print('Ошибкапрогнозапомодели ARIMA: %.3f' % rmse)
plt.figure()
x = np.linspace(X[train_size], X[len(X)-1] , len(X)-train_size)
plt.plot(x, list(test),'b', label = 'Тестовый')
plt.plot(x, predictions, 'r', label = 'Прогноз')
plt.legend()
plt.show()
```

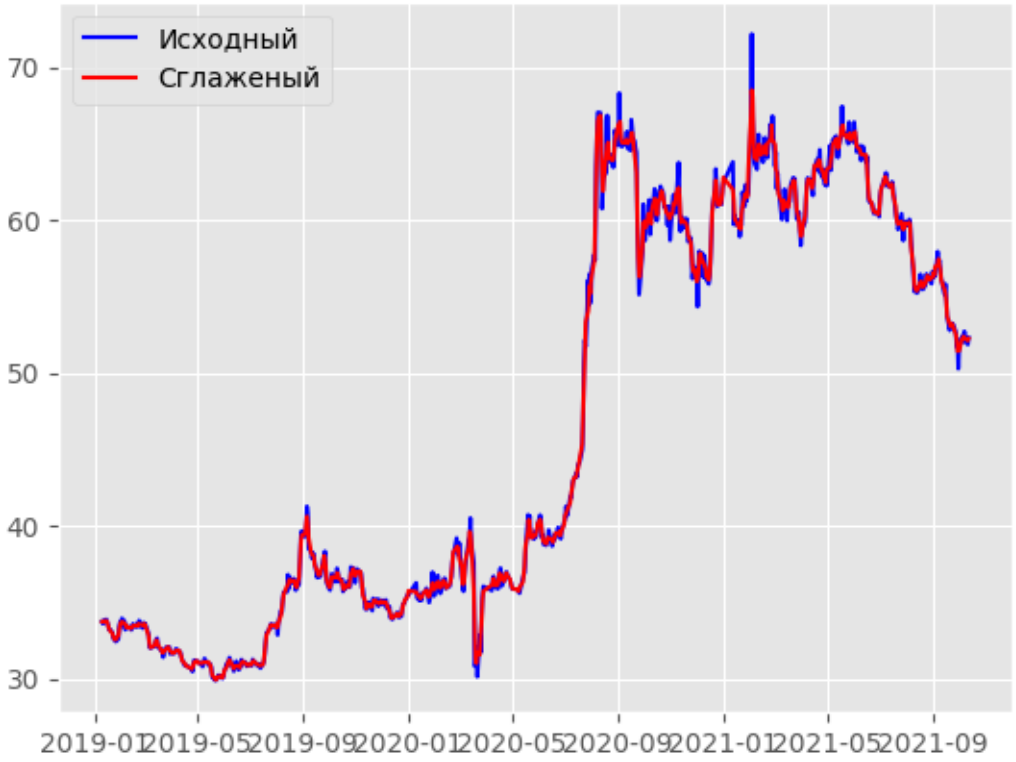


Рис. Исходный график и сглаженный по трем точкам

Практическая работа № 3. Методы машинного обучения для анализа данных

Работа № 3.1 Первичный анализ наборов данных

1. Источник данных: TurkeyStudentEvaluation

http://archive.ics.uci.edu/ml/datasets/Turkiye+Student+Evaluation

Свойства набора данных:

- Набор данных содержит ответы студентов на вопросы о качестве преподавания предметов
- Каждый вопрос оценивается баллами от 1 до 5
- 28 вопросов о качестве преподавания по пройденному предмету
- 3 преподавателя, 13 предметов
- 5820 объектов (записей)

Фрагмент источника данных

instr	class	nb.repeat	attendance	difficulty	Q1	Q2			Q19	Q20
1	2	1	3	5	3	3			3	3
1	2	1	3	4	3	3			3	3
1	2	1	0	1	5	5			5	5
1	2	1	3	5	3	3			3	3
1	2	1	3	4	5	5			5	5

Студенты оценивали параметры difficulty и Q1-Q28. Также приведена информация по посещаемости данным студентом некоторого занятия(attendance), какой раз данный студент проходит данный курс (nb.repeat), номер курса (class) и номер преподавателя (instr).

Описание атрибутов источника данных (на англ. из оригинала, на русском)

Attribute Information:

instr: Instructor's identifier; values taken from {1,2,3}
class: Course code (descriptor); values taken from {1-13}
repeat: Number of times the student is taking this course; values taken from {0,1,2,3,...}
attendance: Code of the level of attendance; values from {0, 1, 2, 3, 4}
difficulty: Level of difficulty of the course as perceived by the student; values taken from {1,2,3,4,5}
Q1: The semester course content, teaching method and evaluation system were provided at the start.
Q2: The course aims and objectives were clearly stated at the beginning of the period.
Q3: The course was worth the amount of credit assigned to it.
Q4: The course was taught according to the syllabus announced on the first day of class.
Q5: The class discussions, homework assignments, applications and studies were satisfactory.
Q6: The textbook and other courses resources were sufficient and up to date.
Q7: The course allowed field work, applications, laboratory, discussion and other studies.
Q8: The quizzes, assignments, projects and exams contributed to helping the learning.
Q9: I greatly enjoyed the class and was eager to actively participate during the lectures.
Q10: My initial expectations about the course were met at the end of the period or year.
Q11: The course was relevant and beneficial to my professional development.
Q12: The course helped me look at life and the world with a new perspective.
Q13: The Instructor's knowledge was relevant and up to date.
Q14: The Instructor came prepared for classes.

Q15: The Instructor taught in accordance with the announced lesson plan.

Q16: The Instructor was committed to the course and was understandable.

Q17: The Instructor arrived on time for classes.

Q18: The Instructor has a smooth and easy to follow delivery/speech.

Q19: The Instructor made effective use of class hours.

Q20: The Instructor explained the course and was eager to be helpful to students.

Q21: The Instructor demonstrated a positive approach to students.

Q22: The Instructor was open and respectful of the views of students about the course.

Q23: The Instructor encouraged participation in the course.

Q24: The Instructor gave relevant homework assignments/projects, and helped/guided students.

Q25: The Instructor responded to questions about the course inside and outside of the course.

Q26: The Instructor's evaluation system (midterm and final questions, projects, assignments, etc.) effectively measured the course objectives.

Q27: The Instructor provided solutions to exams and discussed them with students.

Q28: The Instructor treated all students in a right and objective manner.

Q1-Q28 are all Likert-type, meaning that the values are taken from {1,2,3,4,5}

Информация об полях источника данных:

instr: идентификатор инструктора; значения взяты из {1,2,3}

class: Код курса (дескриптор); значения взяты из {1-13}

repeat: сколько раз студент проходил этот курс; значения взяты из {0,1,2,3, ...}

attendance: Код уровня посещаемости; значения из {0, 1, 2, 3, 4}

difficulty: уровень сложности курса, который воспринимается студентом; значения взяты из {1,2,3,4,5}

Q1: Содержание семестрового курса, метод обучения и система оценивания были предоставлены в начале.

Q2: Цели и задачи курса были четко сформулированы в начале периода.

Q3: Курс стоил присвоенной ему суммы кредита.

Q4: Курс преподавался в соответствии с программой, объявленной в первый день занятий.

Q5: Обсуждения в классе, домашние задания, приложения и исследования были удовлетворительными.

Q6: Учебники и другие ресурсы курсов были достаточными и актуальными.

Q7: Курс допускал полевые работы, приложения, лабораторные, обсуждения и другие исследования.

Q8: Тесты, задания, проекты и экзамены способствовали обучению.

Q9: Мне очень понравился урок, и я очень хотел активно участвовать во время лекций.

Q10: Мои первоначальные ожидания относительно курса оправдались в конце периода или года.

Q11: Курс был актуален и полезен для моего профессионального развития.

Q12: Курс помог мне взглянуть на жизнь и мир с новой точки зрения.

Q13: Знания инструктора были актуальными и актуальными.

Q14: Инструктор прибыл подготовленным к занятиям.

Q15: Инструктор преподавал в соответствии с объявленным планом урока.

Q16: Инструктор был привержен курсу и был понятен.

Q17: Инструктор прибыл вовремя на занятия.

B18: Инструктор легко и четко произносит речь.

Q19: Инструктор эффективно использовал часы занятий.

Q20: Преподаватель объяснил курс и очень хотел помочь студентам.

Q21: Преподаватель продемонстрировал положительный подход к студентам.

Q22: Преподаватель был открыт и уважительно относился к мнению студентов о курсе.

Q23: Инструктор поощрял участие в курсе.

Q24: Преподаватель давал соответствующие домашние задания / проекты и помогал / руководил студентами.

B25: Инструктор ответил на вопросы о курсе внутри и вне курса.

Q26: Система оценки преподавателя (промежуточные и заключительные вопросы, проекты, задания и т. Д.) Эффективно измеряла цели курса.

Q27: Преподаватель предоставил решения к экзаменам и обсудил их со студентами.

Q28: Преподаватель относился ко всем студентам правильно и объективно.

Q1-Q28 все относятся к типу Лайкерта, что означает, что значения взяты из {1,2,3,4,5}

2. Описательные статистики

- Минимум и максимум
- Среднее значение
- Характеристики разброса
- Дисперсия
- Стандартное отклонение
- Интервал изменения
- Квантили
- Матрица ковариаций и корреляций (оценка связи между признаками)

Например: Вычисление описательных статистик NumPy:

```
import numpyas np
data = np.genfromtxt("turkiye.csv", delimiter=',')
for iin range(0,data.shape[1]):
# data.shape[1] размерность
    # второй мерности матрицы
    # (количество столбцов)
print("Признак ",i-2)
x = data[:,i]
    size = np.size(x)
    min = np.min(x)
    max = np.max(x)
print("Минимум: ",min)
print("Максимум: ",max)
    sum = np.sum(x)
print("Сумма: ",sum)
    sum2 = np.dot(x,x)
print("Суммаквадратов: ", sum2)
    mean = sum/size
    mean2 = sum2/size
print("Среднеезначение: ",mean)
print("Момент второго порядка: ", sum2/size)
var = np.var(x)
    SDM = var * size
print("Сумма квадратов отличий от средних: ", SDM)
print("Дисперсия: ",var)
    std = np.sqrt(var)
varcoef = std/mean
print("Среднеквадратичное отклонение: ",std)
print("Коэффициент вариации: ",varcoef)
print(" ")
corr = np.corrcoef(data, rowvar= 0)
print(np.max(corr))
```

```
print("Матрицакорреляции: ", corr)
covar = np.cov(data, rowvar= 0)
print("Матрицаковариаций: ", covar)
x = data[:,0]
print("Квантили: ",np.percentile(x,[25,50,75]))
```

Матрица корреляций признаков сложности предмета и всех вопросов.

	diff	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28
diff	1	0,05	0,07	0,07	0,06	0,06	0,05	0,05	0,05	0,06	0,04	0,06	0,04	0,08	0,09	0,09	0,05	0,12	0,07	0,08	0,09	0,1	0,1	0,08	0,07	0,1	0,06	0,06	0,09
Q1	0,05	1	0,87	0,77	0,85	0,8	0,77	0,79	0,79	0,73	0,8	0,72	0,76	0,72	0,7	0,7	0,74	0,61	0,71	0,7	0,69	0,67	0,67	0,73	0,73	0,67	0,7	0,71	0,66
Q2	0,07	0,87	1	0,85	0,87	0,86	0,83	0,84	0,83	0,8	0,85	0,79	0,8	0,8	0,79	0,79	0,81	0,72	0,79	0,79	0,78	0,76	0,77	0,8	0,8	0,77	0,78	0,77	0,75
Q3	0,07	0,77	0,85	1	0,83	0,84	0,82	0,82	0,81	0,8	0,83	0,81	0,78	0,81	0,81	0,8	0,79	0,77	0,8	0,8	0,8	0,79	0,79	0,8	0,79	0,79	0,8	0,77	0,78
Q4	0,06	0,85	0,87	0,83	1	0,87	0,84	0,84	0,82	0,78	0,84	0,77	0,79	0,78	0,77	0,78	0,79	0,7	0,77	0,77	0,76	0,75	0,75	0,79	0,79	0,75	0,77	0,76	0,74
Q5	0,06	0,8	0,86	0,84	0,87	1	0,88	0,89	0,88	0,81	0,88	0,81	0,82	0,83	0,81	0,81	0,84	0,73	0,82	0,81	0,79	0,78	0,78	0,83	0,83	0,78	0,8	0,79	0,77
Q6	0,05	0,77	0,83	0,82	0,84	0,88	1	0,89	0,86	0,8	0,87	0,8	0,81	0,81	0,8	0,8	0,82	0,72	0,78	0,79	0,78	0,77	0,77	0,8	0,8	0,77	0,79	0,78	0,76
Q7	0,05	0,79	0,84	0,82	0,84	0,89	0,89	1	0,9	0,82	0,89	0,81	0,83	0,81	0,79	0,79	0,82	0,7	0,79	0,79	0,78	0,76	0,76	0,82	0,82	0,77	0,8	0,79	0,75
Q8	0,05	0,79	0,83	0,81	0,82	0,88	0,86	0,9	1	0,83	0,89	0,81	0,84	0,79	0,78	0,77	0,82	0,7	0,79	0,78	0,77	0,75	0,75	0,81	0,82	0,76	0,79	0,79	0,73
Q9	0,06	0,73	0,8	0,8	0,78	0,81	0,8	0,82	0,83	1	0,87	0,83	0,81	0,79	0,79	0,79	0,8	0,75	0,79	0,79	0,78	0,78	0,78	0,79	0,78	0,78	0,78	0,76	0,76
Q10	0,04	0,8	0,85	0,83	0,84	0,88	0,87	0,89	0,89	0,87	1	0,86	0,87	0,84	0,82	0,82	0,86	0,73	0,83	0,82	0,81	0,8	0,8	0,84	0,84	0,79	0,83	0,82	0,78
Q11	0,06	0,72	0,79	0,81	0,77	0,81	0,8	0,81	0,81	0,83	0,86	1	0,86	0,8	0,8	0,8	0,79	0,75	0,78	0,8	0,79	0,79	0,79	0,79	0,79	0,78	0,78	0,77	0,77
Q12	0,04	0,76	0,8	0,78	0,79	0,82	0,81	0,83	0,84	0,81	0,87	0,86	1	0,79	0,77	0,76	0,8	0,69	0,78	0,78	0,76	0,75	0,74	0,79	0,8	0,74	0,77	0,77	0,73
Q13	0,08	0,72	0,8	0,81	0,78	0,83	0,81	0,81	0,79	0,79	0,84	0,8	0,79	1	0,94	0,91	0,9	0,84	0,89	0,88	0,88	0,87	0,87	0,87	0,86	0,87	0,86	0,83	0,86
Q14	0,09	0,7	0,79	0,81	0,77	0,81	0,8	0,79	0,78	0,79	0,82	0,8	0,77	0,94	1	0,93	0,89	0,88	0,89	0,89	0,9	0,89	0,89	0,87	0,86	0,89	0,86	0,83	0,87
Q15	0,09	0,7	0,79	0,8	0,78	0,81	0,8	0,79	0,77	0,79	0,82	0,8	0,76	0,91	0,93	1	0,89	0,88	0,89	0,89	0,89	0,89	0,89	0,88	0,85	0,89	0,86	0,82	0,87
Q16	0,05	0,74	0,81	0,79	0,79	0,84	0,82	0,82	0,82	0,8	0,86	0,79	0,8	0,9	0,89	0,89	1	0,8	0,91	0,88	0,87	0,85	0,85	0,89	0,88	0,85	0,86	0,85	0,83
Q17	0,12	0,61	0,72	0,77	0,7	0,73	0,72	0,7	0,7	0,75	0,73	0,75	0,69	0,84	0,88	0,88	0,8	1	0,85	0,86	0,87	0,87	0,87	0,82	0,79	0,87	0,82	0,77	0,86
Q18	0,07	0,71	0,79	0,8	0,77	0,82	0,78	0,79	0,79	0,79	0,83	0,78	0,78	0,89	0,89	0,89	0,91	0,85	1	0,9	0,88	0,87	0,87	0,88	0,87	0,86	0,86	0,83	0,84
Q19	0,08	0,7	0,79	0,8	0,77	0,81	0,79	0,79	0,78	0,79	0,82	0,8	0,78	0,88	0,89	0,89	0,88	0,86	0,9	1	0,91	0,9	0,89	0,89	0,87	0,88	0,87	0,84	0,86
Q20	0,09	0,69	0,78	0,8	0,76	0,79	0,78	0,78	0,77	0,78	0,81	0,79	0,76	0,88	0,9	0,89	0,87	0,87	0,88	0,91	1	0,93	0,91	0,89	0,86	0,89	0,87	0,83	0,88
Q21	0,1	0,67	0,76	0,79	0,75	0,78	0,77	0,76	0,75	0,78	0,8	0,79	0,75	0,87	0,89	0,89	0,85	0,87	0,87	0,9	0,93	1	0,94	0,89	0,86	0,9	0,87	0,84	0,89
Q22	0,1	0,67	0,77	0,79	0,75	0,78	0,77	0,76	0,75	0,78	0,8	0,79	0,74	0,87	0,89	0,89	0,85	0,87	0,87	0,89	0,91	0,94	1	0,9	0,87	0,91	0,87	0,84	0,89
Q23	0,08	0,73	0,8	0,8	0,79	0,83	0,8	0,82	0,81	0,79	0,84	0,79	0,79	0,87	0,87	0,88	0,89	0,82	0,88	0,89	0,89	0,89	0,9	1	0,92	0,89	0,88	0,87	0,86
Q24	0,07	0,73	0,8	0,79	0,79	0,83	0,8	0,82	0,82	0,78	0,84	0,79	0,8	0,86	0,86	0,85	0,88	0,79	0,87	0,87	0,86	0,86	0,87	0,92	1	0,88	0,88	0,87	0,84
Q25	0,1	0,67	0,77	0,79	0,75	0,78	0,77	0,77	0,76	0,78	0,79	0,78	0,74	0,87	0,89	0,89	0,85	0,87	0,86	0,88	0,89	0,9	0,91	0,89	0,88	1	0,89	0,85	0,9
Q26	0,06	0,7	0,78	0,8	0,77	0,8	0,79	0,8	0,79	0,78	0,83	0,78	0,77	0,86	0,86	0,86	0,86	0,82	0,86	0,87	0,87	0,87	0,87	0,88	0,88	0,89	1	0,88	0,88
Q27	0,06	0,71	0,77	0,77	0,76	0,79	0,78	0,79	0,79	0,76	0,82	0,77	0,77	0,83	0,83	0,82	0,85	0,77	0,83	0,84	0,83	0,84	0,84	0,87	0,87	0,85	0,88	1	0,85
Q28	0,09	0,66	0,75	0,78	0,74	0,77	0,76	0,75	0,73	0,76	0,78	0,77	0,73	0,86	0,87	0,87	0,83	0,86	0,84	0,86	0,88	0,89	0,89	0,86	0,84	0,9	0,88	0,85	1

Рисунок 3.1. Матрица корреляций признаков сложности предмета и всех вопросов

Видно, что вопросы, расположенные рядом обладают как правило большей корреляцией, чем вопросы, расположенные в опроснике дальше друг от друга. Это говорит о том, что опросник составлялся так, чтобы вопросы были расположены по некоторому логическому порядку, возможно группами, которые раскрывают одну из сторон преподавания. Также видно, что сложность фактически не коррелирует с ответами на вопросы.

3. Аномалии в данных

Причины появления аномалий в данных

- Неточности в данных, связанные с неточностью или ошибкой измерительных приборов, отказом оборудования
- Ошибки при сканировании, неточности, связанные с ошибкой распознавания
- Некорректная информация, полученная от людей - опрашиваемых, испытуемых.
- Ошибки при ручном создании наборов данных
  Поиск аномальных объектов
- Работа с пропущенными данными
- Избавление от несогласованности данных, подозрительно выделяющихся значений признаков, работа с выбросами
- Приведение числовых признаков к некоторому стандартному виду

Резюме: Аномалии в данных в разных наборах проявляются по-разному, выработать некоторый одинаковый подход сложно.

Поиск аномальных объектов

- Работа с пропущенными данными

- Избавление от несогласованности данных, подозрительно выделяющихся значений признаков, работа с выбросами
- Приведение числовых признаков к некоторому стандартному виду

Из-за наличия пропусков в данных работа некоторых алгоритмов невозможна. Пути решения – удаление признаков и объектов с большим количеством пропусков, подстановка средних значений по признаку, EM алгоритм подстановки.

В разных местах одни и те же «связующие» данные могут быть записаны в разной форме. Некоторые значения не могут оказаться правдой в силу разных причин (возраст вряд ли может составлять >150, возраст, рост, вес не могут быть отрицательными и для них известно в каких пределах эти данные могут логично располагаться). Некоторые значения могут выбиваться из общего ряда просто потому, что случилось некоторое редкое, маловероятное событие, которое было записано.

Разные числовые признаки могут располагаться в разных диапазонах.

Рассмотрим пример работы с пропущенными данными. Можно заметить, что очень много пропусков у признака 3, объектов 3, 13, 15.

Объект\признак	1	2	3	4	5	Число пропусков	% пропусков
1	1.3	9.9	6.7	3.0	2.6	0	0
2	4.1	5.7			2.9	2	40
3		9.9		3.0		3	60
4	0.9	8.6		2.1	1.8	1	20
5	0.4	8.3		1.2	1.7	1	20
6	1.5	6.7	4.8		2.5	1	20
7	0.2	8.8	4.5	3.0	2.4	0	0
8	2.1	8.0	3.0	3.8	1.4	0	0
9	1.8	7.6		3.2	2.5	1	20
10	4.5	8.0		3.3	2.2	1	20
11	2.5	9.2		3.3	3.9	1	20
12	4.5	6.4	5.3	3.0	2.5	0	0
13					2.7	4	80
14	2.8	6.1	6.4		3.8	1	20
15	3.7			3.0		3	60
16	1.6	6.4	5.0		2.1	1	20
17	0.5	9.2		3.3	2.8	1	20
18	2.8	5.2	5.0		2.7	1	20
19	2.2	6.7		2.6	2.9	1	20
20	1.8	9.0	5.0	2.2	3.0	0	0
число пропусков	2	2	11	6	2	23	
% пропусков	10	10	55	30	10		23

После удаления указанных признаков и объектов остаётся лишь некоторое число пропусков, которые можно попытаться восстановить.

Объект\признак	1	2	4	5	число пропусков	% пропусков
1	1.3	9.9	3.0	2.6	0	0
2	4.1	5.7		2.9	1	25
4	0.9	8.6	2.1	1.8	0	0
5	0.4	8.3	1.2	1.7	0	0
6	1.5	6.7		2.5	1	25
7	0.2	8.8	3.0	2.4	0	0
8	2.1	8.0	3.8	1.4	0	0

Объект\признак	1	2	4	5	число пропусков	% пропусков
9	1.8	7.6	3.2	2.5	0	0
10	4.5	8.0	3.3	2.2	0	0
11	2.5	9.2	3.3	3.9	0	0
12	4.5	6.4	3.0	2.5	0	0
14	2.8	6.1		3.8	1	25
16	1.6	6.4		2.1	1	25
17	0.5	9.2	3.3	2.8	0	0
18	2.8	5.2		2.7	1	25
19	2.2	6.7	2.6	2.9	0	0
20	1.8	9.0	2.2	3.0	0	0
число пропусков	2	2	6	2	5	
% пропусков	0	0	29.4	0		7.35

Пример аномальных данных в TurkeyStudentEvaluation.

Можно заметить большое количество объектов, где все ответы на вопросы одинаковые. Есть некоторые, где все ответы на вопросы кроме оценки сложности одинаковые. В опросниках часто бывает такая ситуация, что человеку лень или нет времени отвечать на вопросы, поэтому он просто ставит одинаковые галочки на все вопросы. Возникает вопрос, какие из объектов действительно являются лишними в смысле их недостоверности. Например, 2 человека не посещали занятия, указали сложность предмета как 1 (очень легкий) и ответили на все вопросы о преподавателе 1 (что в целом говорит об оценке преподавателя как очень плохо): как студенты, не посещавшие занятия, могли оценить преподавателя как плохого во всём? Каждый может придумать свой критерий, какие объекты являются лишними.

Таблица Пример аномальных данных в TurkeyStudentEvaluation

instr	class	nb.repeat	attendance	difficulty	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10		Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28
1	2	1	3	5	3	3	3	3	3	3	3	3	3	3		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
1	2	1	3	4	3	3	3	3	3	3	3	3	3	3		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
1	2	1	0	1	5	5	5	5	5	5	5	5	5	5		5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
1	2	1	3	5	3	3	3	3	3	3	3	3	3	3		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
1	2	1	3	4	5	5	5	5	5	5	5	5	5	5		5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
1	2	1	3	4	2	2	2	2	2	2	2	2	2	2		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1	2	2	1	5	3	3	2	2	5	3	3	3	5	5		4	4	3	4	4	4	4	4	4	4	4	5	4	5	5	4	4	4
1	2	1	2	4	1	1	4	2	3	3	2	2	2	2		3	2	4	3	3	3	5	2	3	3	3	1	3	3	1	3	3	2
1	7	3	0	4	3	3	3	3	3	3	3	3	3	3		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
1	7	1	1	1	1	2	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1
1	7	3	1	3	3	3	3	3	3	3	3	3	3	3		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
1	7	1	0	1	1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	7	1	0	1	1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	7	2	4	5	5	5	5	5	5	5	5	5	5	5		5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
1	7	1	1	3	4	4	4	4	4	4	4	3	3	3		3	4	4	3	3	1	3	2	3	3	3	3	3	3	3	3	3	3
1	7	1	3	3	1	1	5	1	5	4	5	5	1	4		5	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
1	7	1	1	4	4	4	4	4	4	4	4	4	4	4		4	2	4	4	4	4	4	4	4	3	3	4	4	4	4	4	4	4
1	7	1	0	1	5	5	5	5	5	5	5	5	5	5		5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
1	7	1	2	2	3	3	4	4	4	3	4	5	3	3		4	3	4	3	4	3	4	3	4	3	2	3	3	3	4	4	4	3
1	7	1	4	4	4	5	5	4	5	4	4	5	5	5		5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
1	7	1	2	4	2	2	4	2	3	2	3	4	3	3		4	1	4	4	4	3	4	4	4	4	3	2	3	4	4	4	4	3
1	7	1	3	4	4	4	4	4	4	4	4	4	5	5		5	4	4	5	4	4	4	4	4	4	4	4	4	4	4	4	4	4

Методы поиска выбросов

- Поиск выбросов с использованием квартилей (данные выбросы ищутся по одному признаку):
  - Q_1 - значение признака, которое больше 25% значений из данных.
  - Q_3 - значение признака, которое больше 75% значений из данных
  - Выбросом является значение вне интервала $[X_1, X_2]$:

$$X_1 = Q_1 - k \cdot (Q_3 - Q_1) \qquad X_2 = Q_3 - k \cdot (Q_3 - Q_1)$$

- Поиск выбросов по распределениям признаков (данные выбросы могут быть найдены как многомерные – то есть целые объекты):
  - Все объекты, для которых выполнено неравенство, являются выбросами:

$$\sqrt{(x - \bar{x})' \sum_n^{-1} (x - \bar{x})} > g(n, \alpha_n)$$

где Σ – матрица ковариаций признаков.

Стандартизация и нормализация данных

- Стандартизация задаёт чёткие границы, в которых располагаются значения некоторого признака.

  Стандартизация: $a^j = \min_i x_i^j$, $b^j = \max_i x_i^j$

  1) $z_i^j = \frac{2x_i^j - (b^j + a^j)}{b^j - a^j}$, $z_i^j \in [-1,1]$

  2) $z_i^j = \frac{x_i^j - a^j}{b^j - a^j}$, $z_i^j \in [0,1]$
- Нормализация задаёт свойства признака – мат. ожидание 0 и стандартное отклонение 1.

  Нормализация: $\mu^j = \overline{x^j}$, $\sigma_j^2 = \frac{1}{n-1}\sum_i \left(x_i^j - \overline{x^j}\right)^2$

  $\overline{z^j} = 0$, $z_i^j = \frac{x_i^j - \mu^j}{\sigma^j}$, $\frac{1}{n-1}\sum_i \left(z_i^j - \overline{z^j}\right)^2 = 1$

Данные объекты были признаны выбросами (многомерными) среди первых 140 объектов (ответов студентов-посетителей курса №2). Можно заметить, что здесь присутствуют объекты с разными значениями признаков, причём существуют и более «крайние» объекты – из только единиц или пятерок – которые не попали в список выбросов. Применение метода поиска одномерных выбросов также не имеет смысла, поскольку зависит от значения квартилей. В некоторых случаях крайние оценки могут признаваться выбросом, если пользоваться этим методом, что не имеет смысла, поскольку если какая-то из оценок является выбросом, то нет никакого смысла использовать эту оценку при опросах.

Например, если k=1.5, $|Q_1-Q_3| = 2$, то можно показать, что все оценки будут признаны не выбросами. Если $|Q_1-Q_3| = 1$, то будет исключена как минимум 1 оценка из 5 (например, если $Q_1 = 2$, $Q_3 = 3$, то будет исключена оценка 5, если $Q_1 = 4$, $Q_3 = 5$, будут исключены 1 и 2). Если $|Q_1-Q_3| = 0$, будут исключены все оценки кроме $Q_1 = Q_3 =$ оценка.

Резюме:

- Ковариационная матрица близка к вырожденной (определитель ~0)
- Объекты в большинстве либо очень далеки от того, чтобы быть выбросами, либо выбросы при практически любом уровне значимости
- Объекты-выбросы практически не меняются при разумном изменении параметра уровня значимости
- Объекты, которые были сочтены выбросами не выглядят аномальными
- В данном случае анализ многомерных выбросов не имеет смысла. Необходимо придумать критерий удаления аномальных объектов.

Таблица. Пример объектов - выбросов базы TurkeyEvaluationStudent

i	c	nb	att	Diff	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	2	1	2	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	3	3	3	2	2	1	1	1	1
1	2	1	3	4	5	5	4	4	5	5	4	4	5	5	5	4	5	5	4	4	5	5	5	4	4	5	5	4	4	4	5	4
1	2	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	1	1	1	3	2	2	2	2
1	2	1	2	4	5	3	3	3	2	2	3	3	3	4	4	5	5	4	3	3	3	4	2	2	4	4	5	5	4	4	5	5
1	2	1	1	2	1	1	1	1	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
1	2	1	3	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3
1	2	1	2	3	1	1	1	1	2	2	2	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1	2	1	3	4	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	3	3	3	3	3	3	4	4	2	4	2	1	3
1	2	1	1	3	4	4	4	4	4	4	5	5	5	5	5	4	4	5	4	4	4	4	4	4	4	4	4	4	4	5	4	4
1	2	2	1	3	2	3	3	3	2	5	5	5	5	5	5	5	3	3	3	3	3	3	3	3	3	3	3	2	2	1	1	1
1	2	1	3	4	2	3	4	5	5	4	4	4	5	4	4	4	4	4	4	4	4	2	2	2	4	2	2	4	2	2	3	2
1	2	1	1	3	4	4	4	3	4	2	4	5	3	3	4	1	5	5	5	5	5	5	5	5	5	5	3	4	5	4	4	5
1	2	1	1	1	1	1	1	1	1	1	1	5	1	1	1	5	5	5	5	5	5	4	5	5	5	5	5	5	5	4	5	5
1	2	1	3	3	2	4	4	2	5	5	5	5	4	4	5	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	5
1	2	1	4	3	3	5	4	4	5	5	4	5	5	5	5	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
1	2	1	3	3	4	4	3	3	3	3	3	3	3	3	3	5	4	4	3	3	4	3	3	3	3	3	3	3	3	3	3	3
1	2	1	4	3	4	4	4	3	4	3	4	4	4	4	4	4	5	5	5	5	5	5	4	4	4	4	4	4	4	4	4	4
1	2	1	0	1	3	3	1	3	1	2	2	2	2	1	1	1	3	4	4	3	2	4	1	3	3	3	2	3	4	2	3	3
1	2	1	1	5	5	2	2	2	5	5	5	5	5	5	5	5	5	5	5	5	4	5	5	5	5	5	5	4	5	5	1	5
1	2	1	1	4	3	3	3	4	4	4	4	3	3	4	4	4	4	3	3	3	4	4	4	4	4	4	4	4	4	4	4	4
1	2	1	3	4	3	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	4	4	4	4	1	1	3	3	3	3	3	3

Практическое задание

1. Предложить методы анализа выбросов, учитывая особенности данных. Сделать анализ выбросов, удалить выбросы.
2. Проанализировать матрицу корреляций оценок по различным критериям качества преподавания. Выявить значимые корреляции. Объяснить высокие и низкие корреляции.
3. Сравнить матрицы корреляций для разных предметов.
4. Проанализировать описательные статистики по преподавателям, разработать метод сравнения преподавателей по приведённым данным.
5. Проанализировать описательные статистики по предметам, разработать метод сравнения предметов по данным из набора.

Контрольные вопросы:

- Как можно привести данные к единообразному виду?
- Какие есть инструменты для работы с данными?
- Какие простые метрики можно использовать для работы с данными?
- Как можно очистить данные от ненужных/мешающих элементов?
- Как работать с конкретными данными?
- Какие объекты можно признать аномальными в базе TurkeyStudentEvaluation?
- Какую информацию можно извлечь из данных?
- Как можно использовать эту информацию в будущем?

Работа № 3.2 Линейная регрессия

Цель работы: получение практических навыков построения и использования регрессионных моделей на языке Python с использованием библиотек Scikit-Learn и StatsModels.

Задание: используя программу Jupiter Notebook, язык программирования Python, библиотеки Scikit-Learn, StatsModels, NumPy, Matplotlib и др. выполнить следующие задания:

- **Парная регрессия**: построить две реализации парной линейной регрессионной модели на базе двух библиотек Scikit-Learn, StatsModels, сравнить и интерпретировать полученные результаты, входные данные рассчитать согласно варианту в таблице.
- **Множественная регрессия**: для своего варианта провести регрессионное моделирование (построить множественную регрессионную модель, ссылка для скачки данных на странице в разделе Data tables, выбрать не менее 50 строк):
  - выбрать выходную прогнозируемую переменную,
  - построить регрессионную модель со значимыми параметрами (оценить параметры межфакторной корреляции, последовательно добавлять факторы и сравнивать качество получаемых моделей, подобрать вид функции (визуальный анализ), оценить адекватность модели по статистическим показателям, каждый из этапов прокомментировать в отчете),
  - интерпретируете результаты моделирования (что значит полученная формула, какие переменные вносят больший вклад, что будет при изменении независимых переменных с зависимой),
  - прогнозировать новые значения с помощью построенной модели.

1. Теоретические сведения

Регрессия и классификация являются задачами машинного обучения с учителем. Обе используют сходную концепцию использования известных наборов данных, при этом используется алгоритм для изучения функции отображения входной переменной (x) в выходную переменную, то есть y=f(x). У задач есть общие черты и различия (Таблица).

Таблица – Сопоставление задач регрессии и классификации

	Регрессия	**Классификация**
Вид обучения	Обучение с учителем	
Задача	максимально точно аппроксимировать функцию отображения (f), чтобы при появлении новых входных данных (x) можно было прогнозировать выходные данные (y)	
Выходное значение	*числовые* или *непрерывные* (действительные числа)	*дискретные* или *категориальные*
Примеры алгоритмов	линейная регрессия, регрессия в алгоритмах опорных векторов SVR (support vector regression) и регрессионные деревья	логистическая регрессия, наивный байесовский алгоритм, решающие деревья и k-NN (k ближайших соседей)

Линейная регрессия (Linear regression) — один из самых фундаментальных алгоритмов, используемых для моделирования отношений между зависимой переменной и несколькими независимыми переменными. Целью обучения является поиск линии наилучшего соответствия.

Например: $\hat{y} = b_0 + b_1 x$ – уравнение линейной регрессии, которое описывает прямую, наиболее точно показывающую взаимосвязь между входными переменными **x** и выходными переменными **y**. Для составления этого уравнения нужно найти коэффициенты **b** для входных переменных.

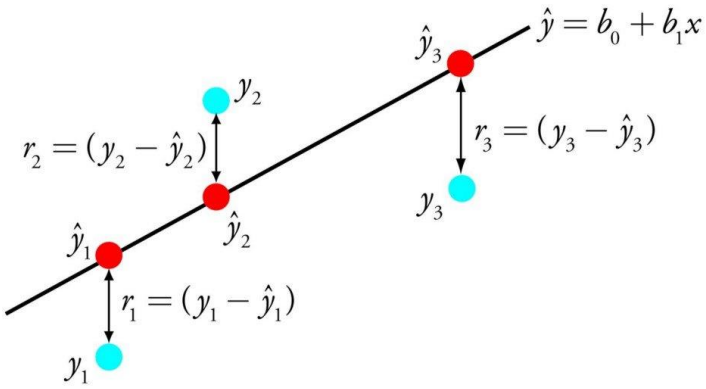


Рис. Графическое представление метода наименьших квадратов

В процессе обучения линейной регрессии находится минимизация квадратов расстояний между точками и линией наилучшего соответствия. Этот процесс известен как минимизация суммы квадратов остатков. Остаток равен разности между предсказанным значением и реальным.

$$J = \min_{b_i} \sum_{i=1}^{n} (y_i - \hat{y_i}))^2 = \min_{b_i} \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_i))^2$$

Для оценки регрессионной модели используются различные методы вроде линейной алгебры или метода наименьших квадратов.

2. Задача

Рассмотрим задачу прогнозирования цен на рынке недвижимости.

- Какими данными о недвижимости мы можем располагать?
- Какие признаки влияют на цену?

Характеристики объектов недвижимости

1. Объективные характеристики:
   - *технический паспорт*
2. Субъективные характеристики (Как измерить?):
   - *состояние объекта недвижимости;*
   - *престижность района;*
   - *…*

Набор данных kc_house_data

https://www.kaggle.com/harlfoxem/housesalesprediction

Рассмотрим задачу прогнозирование цен на примере набора данных kc_house_data.

kc_house_data содержит данные о продажах индивидуальных домов в период с мая 2014 года по май 2015 в округе Кинг, штат Вашингтон, США.

Название признака	Описание
id	уникальный идентификационный номер проданного дома
date	дата продажи дома
bedrooms	количество спален
bathrooms	количество ванных комнат (где 0.25 обозначает, что комната с туалетом, 0.5 – комната с туалетом и раковиной)
sqft_living	общая площадь дома

sqft_lot	площадь прилегающей территории
floors	количество этажей
waterfront	бинарный атрибут, указывающий на то, есть ли вид на реку или нет
view	оценка внешнего вида дома (от 0 до 4)
condition	оценка состояния дома (от 0 до 5)
grade	оценка качества строительства и дизайна здания (от 1 до 13)
sqft_above	общая площадь наземной части дома
sqft_basement	общая площадь подземного части дома
yr_built	год строительства дома
yr_renovated	год последнего ремонта или последней реконструкции
zipcode	почтовый индекс дома
lat	широта
long	долгота
sqft_lot15	средняя общая площадь 15 ближайших домов
sqft_lot15	средняя площадь прилегающей территории 15 ближайших домов

3. Предобработка данных

Возможно ли уменьшить количество признаков?

Атрибуты *id, date, zipcode, lat, long.*

Удаляем *id, date, zipcode, lat, long* и sqft_basement (sqft_basement = sqft_living - sqft_above).

Пропуски в данных? Нет

Выбросы?

Формат данных: csv файл с разделителем в виде запятой. Используем библиотеку pandas.

```
import pandas as pd
from sklearn.model_selection import train_test_split

data = pd.read_csv("kc_house_data.csv", parse_dates= ['date']) # load the
data into a pandas dataframe
print(data.shape)
data.drop(['id', 'date','sqft_basement','lat','long'], axis = 1, inplace=
True)
print(data.shape)
for column in data:
print(column,end=';')
print()
# Формирование обучающей и тестовой выборок
train_data, test_data = train_test_split(data, train_size= 0.8,
random_state= 60)
train_data.astype(float).to_csv('kc_house_train_data.csv',sep=',',index=Fa
lse, header=False)
test_data.astype(float).to_csv('kc_house_test_data.csv',sep=',',index=Fals
e, header=False)
```

4. Модели регрессии

4.1. Линейная регрессия

Модель **линейной регрессии** имеет вид:

$price = w_0 + \sum_{j=1}^{m} w_j x^j + \varepsilon,$

где x^j - значение признака j.

Задача: найти коэффициенты w наиболее точно предсказывающие цены на имеющихся данных. Как измерить ошибку?

$$\varepsilon_i = price_i - w_0 - \sum_{j=1}^{m} w_j x_i^j$$

Определение **коэффициентов регрессии**

Введем обозначения:

$$\tilde{X} = (\tilde{x}_{ij}) = \begin{pmatrix} 1 & x_1^1 & \dots & x_1^m \\ \dots & \dots & \dots & \dots \\ 1 & x_n^1 & \dots & x_n^m \end{pmatrix}, w = \begin{pmatrix} w_0 \\ \dots \\ w_m \end{pmatrix}, y = \begin{pmatrix} y_1 \\ \dots \\ y_n \end{pmatrix}$$

В случае, когда матрица $\tilde{X}^T\tilde{X}$ невырождена, система нормальных уравнений имеет единственное решение:

$$\widehat{w} = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T y$$

Каждый коэффициент регрессии отражает влияние конкретного признака на цену.

Если матрица $\tilde{X}^T\tilde{X}$ вырождена, система может не иметь решений или иметь бесконечное множество решений. Нет возможности интерпретировать коэффициенты регрессии.

Как быть, когда матрица вырождена или близка к вырожденной (плохо обусловленные или некорректные задачи)?

Значимые коэффициенты регрессии

Прогнозируемая переменная $Y = w_0 + \sum_{j=1}^{m} w_j x^j + \varepsilon,\ \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon)$, а коэффициенты регрессии, найденные с помощью МНК, равны $\widehat{w_0}, \dots, \widehat{w_m}$,. Как понять, что $w_j \neq 0$?

Z-score:

$$z_i = \frac{\widehat{w_i}}{\hat{\sigma}\sqrt{v_i}},\ \forall i = \overline{0, m}$$

где $\hat{\sigma} = \sqrt{\frac{1}{n-m-1}\sum_{i=1}^{n}(y_i - \widehat{w_0} - \sum_{j=1}^{m}\widehat{w_j}x^j)^2}$, v_i - диагональный элемент $(\tilde{X}^T\tilde{X})^{-1}$.

Если принять за нулевую гипотезу, что $w_j = 0$, то:

$$z_i \sim t_{n-m-1}$$

Чем больше абсолютное значение z_i, тем больше уверенность, что $w_j \neq 0$.

Модель линейной регрессия в Scikit-learn

```
from sklearn import linear_model

def linear_regression_model(dataX, dataY):
# Создать линейную регрессионную модель normalize=True
regr = linear_model.LinearRegression()
regr.fit(dataX, dataY) # Обучениемодели
return regr
```

Прогнозирование (Scikit-learn)

```
regressionModel.predict(data)
```

Коэффициент детерминации

Коэффициент детерминации. Коэффициент детерминации определяется следующим образом:

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

где $\bar{y}$ - среднее значение по наблюдаемым данным, $\widehat{y_i}$ - значения объясняемой переменной, рассчитанные с помощью функции регрессии.

Коэффициент детерминации принимает значение от 0 до 1.

Для нахождения R^2 необходимо вызвать функцию r2_score

```
from sklearn.metrics import r2_score
r2_score (trueY, predictedY)
```

Среднеквадратичная ошибка

Для нахождения среднеквадратичной ошибки определим функцию *RMSE*.

Команда *mean_squared_error*, возвращает средний квадрат ошибки (MSE).

```
from sklearn.metrics import mean_squared_error
def RMSE(trueY, predictedY):
    return np.sqrt(mean_squared_error(trueY, predictedY))
```

Качество регрессии (kc_house_data)

- Как влияет формат данных на качество регрессии?
- Действительно ли стоимость дома линейно зависит от признаков bedrooms, bathrooms, floors, view, condition, grade?

Будем считать эти признаки категориальными и бинаризуем их.

```
categorial_cols = ['floors', 'view', 'condition', 'grade', 'bedrooms',
'bathrooms','zipcode']
for cc in categorial_cols:
    dummies = pd.get_dummies(data[cc], drop_first=False)
    dummies = dummies.add_prefix("{}#".format(cc))
data.drop(cc, axis=1, inplace=True)
    data = data.join(dummies)
print(data.shape)
for column in data:
print(column,end=';')
print()
```

В результате был получен набор с 79 признаками.

4.2. Гребневая регрессия

Гребневая регрессия также является линейной моделью регрессии, поэтому ее формула аналогична той, что используется в обычном методе наименьших квадратов. В гребневой регрессии коэффициенты (w) выбираются не только с точки зрения того, насколько хорошо они позволяют предсказывать на обучающих данных, они еще подгоняются в соответствии с дополнительным ограничением. Нам нужно, чтобы величина коэффициентов была как можно меньше. Другими словами, все элементы w должны быть близки к нулю. Это означает, что каждый признак должен иметь как можно меньшее влияние на результат (то есть каждый признак должен иметь небольшой регрессионный коэффициент) и в то же время он должен по-прежнему обладать хорошей прогнозной силой. Это ограничение является примером регуляризации (regularization). Регуляризация означает явное ограничение модели для предотвращения переобучения. Регуляризация, использующаяся в гребневой регрессии, известна как L2 регуляризация.

Метод гребневой регрессии решает проблему вырожденности с помощью добавления к функционалу Q регуляризатора, штрафующего большие значения квадрата евклидовой нормы вектора w:

$$Q_{\text{гр}}(w_0, \dots, w_m) := \left\|\tilde{X}w - y\right\|_2^2 + \alpha\|w\|_2^2 \to \min_w$$

где $\alpha \geq 0$

Для дифференцируемой функции $Q_{\text{гр}}$ необходимое условие минимума $\frac{\partial Q_{\text{гр}}}{\partial w_j} = 0, j = 0,1,\dots,m$ в матричной форме имеет вид:

$$2\tilde{X}^T\left(\tilde{X}\hat{w} - y\right) + 2\alpha I_n\hat{w} = 0 \Longrightarrow (\tilde{X}^T\tilde{X} + \alpha I_n)\hat{w} = \tilde{X}^T y$$

Откуда получаем, что, в случае невырожденности матрицы $\tilde{X}^T\tilde{X} + \alpha I_n$, решение системы нормальных уравнений выглядит следующим образом:

$$\hat{w} = (\tilde{X}^T\tilde{X} + \alpha I_n)^{-1}\tilde{X}^T y$$

Гребневая регрессии (Scikit-learn)

Гребневая регрессии реализована в классе linear_model.Ridge.

```
def ridge_regression_model(dataX, dataY, alphaParam):
# Инициализациямодели
ridge = linear_model.Ridge(alpha= alphaParam)
ridge.fit(dataX, dataY) # Обучение модели
return ridge
```

Модель Ridge позволяет найти компромисс между простотой модели (получением коэффициентов, близких к нулю) и качеством ее работы на обучающем наборе. Компромисс между простотой модели и качеством работы на обучающем наборе может быть задан пользователем при помощи параметра alpha. Оптимальное значение alpha зависит от конкретного используемого набора данных. Увеличение alpha заставляет коэффициенты сжиматься до близких к нулю значений, что снижает качество работы модели на обучающем наборе, но может улучшить ее обобщающую способность.

4.3. Лассо-регрессия

Альтернативой Ridge как метода регуляризации линейной регрессии является Lasso (англ.Least Absolute Shrinkage and Selection Operator, Оператор наименьшего сжатияи выбора). Как и гребневая регрессия, лассо также сжимает коэффициенты до близких к нулю значений, но несколько иным способом, называемым L1 регуляризацией. Результат L1 регуляризации заключается в том, что при использовании Лассо некоторые коэффициенты становятся равны точно нулю. Получается, что некоторые признаки полностью исключаются из модели. Это можно рассматривать как один из видов автоматического отбора признаков. Получение нулевых значений для некоторых коэффициентов часто упрощает интерпретацию модели и может выявить наиболее важные признаки вашей модели.

Метод Лассо штрафует большие значения L_1- нормы вектора w:

$$Q_{\text{л}}(w_0,\dots,w_m) \coloneqq \left\|\tilde{X}w - y\right\|_2^2 + \beta\|w\|_1 \to \min_w$$

где $\beta \geq 0$

Свойство:

При увеличении β количество коэффициентов регрессии равных нулю увеличивается, то есть происходит отбор значимых признаков (featureselection).

Функция $Q_{\text{л}}$ не является дифференцируемой функцией на всем множестве $\mathbb{R}^{m+1}$, но является субдифференцируемой, поэтому воспользуемся необходимым условием минимума в терминах субдифференциала:

$$0 \in \partial_{w_j} Q_{\text{л}}, j = 0,1,\dots,m$$

Подробно распишем чему равен $\partial_{w_j} Q_{\text{л}}$:

$$\partial_{w_j} Q_л = \frac{\partial Q}{\partial w_j} + \beta \partial_{w_j} |w_j| = -2\rho_j + 2z_j w_j + \begin{cases} -\beta, \text{ при } w_j < 0 \\ [-\beta, \beta], \text{ при } w_j = 0 \\ \beta, \text{ при } w_j < 0 \end{cases}$$

$$= \begin{cases} 2z_j w_j - 2\rho_j - \beta, \text{ при } w_j < 0 \\ [-2\rho_j - \beta, -2\rho_j + \beta], \text{ при } w_j = 0 \\ 2z_j w_j - 2\rho_j + \beta, \text{ при } w_j < 0 \end{cases}$$

где $\rho_j = \sum_{i=1}^{n} \tilde{x}_{ij}\left(y_i - \sum_{k=0.k\neq j}^{m} w_k \tilde{x}_{ik}\right), z_j = \sum_{i=1}^{n} \left(\tilde{x}_{ij}\right)^2$

Получаем, что $\hat{w}_j = \begin{cases} (\rho_j + \beta/2)/z_j, \text{ при } \rho_j < -\beta/2 \\ 0, \text{ при } \rho_j \in [-\beta/2, \beta/2] \\ (\rho_j - \beta/2)/z_j, \text{ при } \rho_j > \beta/2 \end{cases}$

В Scikit-learn для нахождения коэффициентов регрессии методом Лассо используется следующий функционал качества:

$$\frac{1}{2n} \left\| \tilde{X} w - y \right\|_2^2 + \alpha \|w\|_1 \to \min_w$$

где$\alpha \geq 0$

Лассо (Scikit-*learn*)

```
def lasso_model(dataX, dataY, alphaParam):
# Инициализациямодели
lasso = linear_model.Lasso(alpha=alphaParam, max_iter=1000)
lasso.fit(dataX, dataY)  # Обучение модели
return lasso
```

На практике, когда стоит выбор между гребневой регрессией и лассо, предпочтение, как правило, отдается гребневой регрессии. Однако, если у вас есть большое количество признаков и есть основания считать, что лишь некоторые из них важны, Lasso может быть оптимальным выбором. Аналогично, если вам нужна легко интерпретируемая модель, Lasso поможет получить такую модель, так как она выберет лишь подмножество входных признаков.

5. Практическое задание

1. Выполните предобработку данных (preprocessing):
   а) Анализ и удаление выбросов
   б) Анализ и восстановление пропусков
   в) Стандартизация данных
   г) Обсудить возможность выделения характерных признаков (featureextraction).
2. Выполнить вычисления по модели многомерной линейной регрессии и провести анализ полученной модели:
   а) Оценить качество регрессии по коэффициенту детерминации
   б) Оценить ошибку RMSE
   в) Выделить значимые и незначимые коэффициенты регрессии

Качество регрессии

	RMSE	R^2
Линейная регрессия		
Гребневая регрессия($\alpha = 3$)		
Лассо ($\alpha = 120$)		

Коэффициенты регрессии

	Линейная регрессия	Гребневая регрессия ($\alpha = 3$)	Лассо ($\alpha = 120$)
sqft_living			
sqft_lot			
waterfront			
sqft_above			
yr_built			
yr_renovated			
sqft_living15			
sqft_lot15			
floors#1.0			
floors#1.5			
floors#2.0			
floors#2.5			
floors#3.0			
floors#3.5			

Работа № 3.3 Анализ главных компонент

1. Теоретические сведения

Это один из основных алгоритмов машинного обучения. Позволяет уменьшить размерность данных, потеряв наименьшее количество информации. Вычисление главных компонент сводится к вычислению собственных векторов и собственных значений ковариационной матрицы исходных данных или к сингулярному разложению матрицы данных.

Метод главных компонент (МГК, англ. principal component analysis, PCA) — это техника машинного обучения, которая используется для изучения взаимосвязей между наборами переменных. Другими словами, МГК изучает наборы переменных для того, чтобы определить базовую структуру этих переменных. МГК еще иногда называют факторным анализом.

Основные приложения

- Dimensionality reduction. Снижение размерности данных при сохранении всей или большей части информации

Метод главных компонент используется для преобразования набора данных с множеством параметров в новый набор данных с меньшим количеством параметров и каждый новый параметр этого набора данных — это линейная комбинация раннее существующих параметров. Эти преобразованные данные стремятся обосновать большую часть дисперсии оригинального набора данных с гораздо большей простотой.

- Feature extraction. Выявление и интерпретация скрытых признаков

Нередко в машинном обучении встречаются ситуации, когда данные собираются априори, и лишь затем возникает необходимость разделить некоторую выборку по известным классам. Как следствие часто может возникнуть ситуация, когда имеющийся набор признаков плохо подходит для эффективной классификации. По крайней мере, при первом приближении.

В такой ситуации можно строить композиции слабо работающих по отдельности методов, а можно начать с обогащения данных путём выявления скрытых зависимостей между признаками. И затем строить на основе найденных зависимостей новые наборы признаков, некоторые из которых могут потенциально дать существенный прирост качества классификации.

Различия линейной регрессии и МГК

Линейная регрессия определяет линию наилучшего соответствия через набор данных. Метод главных компонентов определяет несколько ортогональных линий наилучшего соответствия для набора данных.

2. Задача: проанализировать заемщиков банка на основе различных данных

Пример источника данных: GiveMeSomeCredit
https://www.kaggle.com/c/GiveMeSomeCredit

Variable Name	Description	Type
RevolvingUtilizationOfUnsecuredLines	Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits Общий баланс по кредитным картам и личным кредитным линиям, за исключением долга по недвижимости	percentage

	и без рассрочки, например автокредитов, деленный на сумму кредитных лимитов	
Age	Age of borrower in years Возрастзаемщика в годах	integer
NumberOfTime30-59DaysPastDueNotWorse	Number of times borrower has been 30-59 days past due but no worse in the last 2 years. Количество просроченных платежей заемщика на 30-59 дней, но не больше чем за последние 2 года.	integer
DebtRatio	Monthly debt payments, alimony, living costs divided by monthy gross income Ежемесячные выплаты по долгу, алименты, расходы на жизнь, разделенные на ежемесячный валовой доход	percentage
MonthlyIncome	Monthly income Ежемесячный доход	real
NumberOfOpenCreditLinesAndLoans	Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards) Количество открытых займов (рассрочка, например, автокредит или ипотека) и кредитных линий (например, кредитные карты)	integer
NumberOfTimes90DaysLate	Number of times borrower has been 90 days or more past due. Количество просроченных платежей заемщика на 90 дней или более.	integer
NumberRealEstateLoansOrLines	Number of mortgage and real estate loans including home equity lines of credit Количество ипотечных кредитов и ссуд на недвижимость, включая кредитные линии под залог собственного капитала	integer
NumberOfTime60-89DaysPastDueNotWorse	Number of times borrower has been 60-89 days past due but no worse in the last 2 years. Количество раз, когда заемщик просрочил платеж на 60-89 дней, но не больше чем за последние 2 года.	integer
NumberOfDependents	Number of dependents in family excluding themselves (spouse, children etc.) Количество иждивенцев в семье, исключая их самих (супруга, дети и т. д.)	integer

Пример: Give Me Some Credit

Revolving Utilization Of Unsecured Lines	age	Number Of Time 30-59 Days Past Due Not Worse	Debt Ratio	Monthly Income	Number Of Open Credit Lines And Loans	Number Of Times 90 Days Late	Number Real Estate Loans Or Lines	Number Of Time 60-89 Days Past Due Not Worse	Number Of Dependents
0.766126609	45	2	0.802982129	9120	13	0	6	0	2
0.957151019	40	0	0.121876201	2600	4	0	0	0	1
0.65818014	38	1	0.085113375	3042	2	1	0	0	0
0.233809776	30	0	0.036049682	3300	5	0	0	0	0
0.9072394	49	1	0.024925695	63588	7	0	1	0	0
0.213178682	74	0	0.375606969	3500	3	0	1	0	1
0.305682465	57	0	5710	NA	8	0	3	0	0
0.754463648	39	0	0.209940017	3500	8	0	0	0	0
0.116950644	27	0	46	NA	2	0	0	0	NA
0.189169052	57	0	0.606290901	23684	9	0	4	0	2
0.644225962	30	0	0.30947621	2500	5	0	0	0	0
0.01879812	51	0	0.53152876	6501	7	0	2	0	2
0.010351857	46	0	0.298354075	12454	13	0	2	0	2
0.964672555	40	3	0.382964747	13700	9	3	1	1	2

3. Задача снижения размерности

Представить набор данных меньшим числом признаков таким образом, чтобы потеря информации, содержащейся в оригинальных данных, была минимальной.

Принципы компонентного анализа

Данные заданы матрицей $X = (x_i^j)$ размерности $n \times m$, где $i = \overline{1,n}$ и $j = \overline{1,m}$, n – число наблюдений (объектов), m – число признаков.

Обозначим за C $(m \times m)$ матрицу ковариаций признаков матрицы X:

$$c_{ij} = \frac{\sum_{p=1}^{n} x_k^i x_k^j}{n} - \mu_i \mu_j, \forall i,j \in \{1 \dots m\},$$

$$\mu_i - среднее значение признака i, i \in \{1 \dots m\}$$

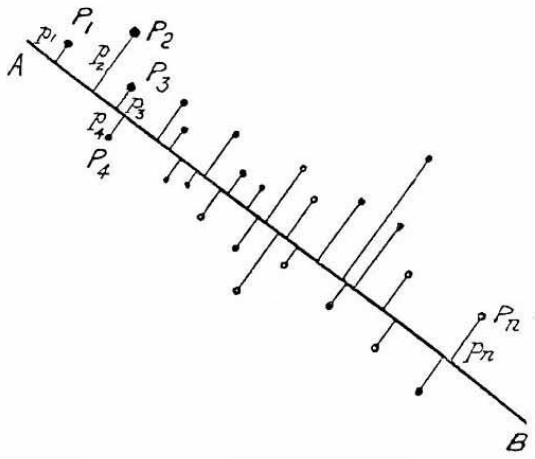
В матричном виде:

$$C = \frac{X^T X}{n} - \mu^T \mu, \mu = (\mu_1 \dots \mu_m)$$

Вариация i-го признака: $Var(x^i) = c_{ii}$

Общая вариация данных: $Var(X) = \sum_{i=1}^{m} c_{ii}$

Задача: найти ортогональные векторы такие, что $v^T C v \to max,$ т.е. проекция данных, на которые позволит сохранить наибольшую вариацию



Матрица C симметричная и положительно определена. Имеет место равенство:

$$C = V \Lambda V^T$$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_m \end{bmatrix},$$

$$\lambda - собственные значения матрицы C, \sum_{i=1}^{m} \lambda_i = \sum_{i=1}^{m} c_{ii}, \lambda_1 > \lambda_2 \geq \dots \geq \lambda_m \geq 0$$

$$V(m \times m) - матрица собственных векторов матрицы C$$

Главные компоненты:

$$U = X \cdot \lfloor v^1, v^2, \dots, v^k \rfloor^T, k < m$$

Доля объясненной вариации:

$$\frac{\sum_{i=1}^{k} \lambda_i}{Var(X)}$$

Singular value decomposition

- Данные заданы матрицей $X = (x_i^j)$ размерности $n \times m$, где $i = \overline{1,n}$ и $j = \overline{1,m}$, n – число наблюдений (объектов), m – число признаков.
- Требуется среди всех матриц такого же размера $n \times m$и ранга $\leq k$найти матрицу Y, для которой норма матрицы $\|X - Y\|$ будет минимальной.
- Решение зависит от матричной нормы
- Наиболее подходящие: Евклидова норма и норма Фробениуса:
    - Евклидова норма: $\|A\|_2 = \sqrt{\lambda T_{max}}$, где λ_{max} - максимальное собственное значение матрицы A
    - Норма Фробениуса: $\|A\|_F = \sqrt{\sum_i \sum_i a_{ij}^2}$

Существуют такие матрицы Uи V, что выполняется равенство $X = U \cdot S \cdot V^T$,

где U – матрица собственных векторов матрицы $X \cdot X^T$, V – матрица собственных векторов матрицы $X^T \cdot X$, а матрица S размерности $n \times m$имеет на главной диагонали элементы $\sigma_1, \sigma_2, \ldots, \sigma_m$ и все остальные нули, где σ_i - сингулярные числа матрицы X, а σ_i^2 - собственные числа матрицы $X^T \cdot X$.

Запишем матрицы Uи V в векторном виде:

$$U = \lfloor u^1, u^2, \ldots, u^n \rfloor, \rightleftarrows V = \lfloor v^1, v^2, \ldots, v^m \rfloor$$

Тогда SVD разложение можно представить как

$$X = \sigma_1 u^1 (v^1)^T + \sigma_2 u^2 (v^2)^T + \cdots + \sigma_m u^m (v^m)^T$$

Теорема Шмидта-Мирского:

Решением матричной задачи наилучшей аппроксимации в норме Евклида и в норме Фробениуса является матрица $X^* = \sigma_1 u^1 (v^1)^T + \sigma_2 u^2 (v^2)^T + \cdots + \sigma_k u^k (v^k)^T$

Ошибки аппроксимации:

$$\|X - X^*\|_2 = \sigma_{k+1}$$

$$\|X - X^*\|_F = \sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots \sigma_m^2}$$

Выбор числа k главных факторов

Общая вариация данных:

$$Var(X) = \sigma_1^2 + \sigma_2^2 + \ldots + \sigma_m^2$$

Доля объясненной вариации:

$$\frac{\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_k^2}{Var(X)}, k < m$$

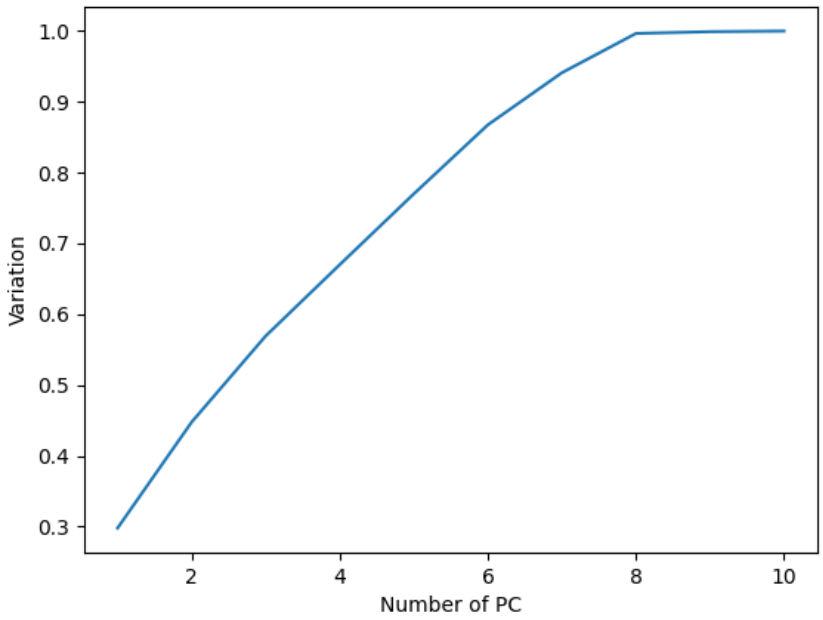
Хорошим значением считается доля объясненной вариации ≥ 80%

4. Решениевscikit-learn

```
import numpyas np
import scipyas sp
from sklearn.decompositionimport PCA
import matplotlib.pyplotas plt
from sklearn.preprocessingimport scale

np.set_printoptions(precision=10,
threshold= 10000,
suppress= True)
```

```
# Загружаем данные и удаляем наблюдения с пропущенными значениями
data = np.genfromtxt("cs-data.csv", delimiter = ',',
skip_header= 1, usecols=list(range(1, 11)))
data = data[~np.isnan(data).any(axis = 1)]

# Выполняем метод главных компонент
data = scale(data)
pca = PCA(svd_solver='full')
pca.fit(data)

print("Размерность данных \n", data.shape, "\n")
# Вклад каждого фактора в объяснение вариации
print("Вклад каждого фактора в объяснение вариации \n",
pca.explained_variance_ratio_, "\n")
# Рост доли объясненной вариации с увеличением числа главных факторов
var = np.round(np.cumsum(pca.explained_variance_ratio_), decimals=4)
print("Рост доли объясненной вариации с увеличением числа главных факторов
\n", var, "\n")
plt.figure()
plt.plot(np.arange(1,11), var)
plt.ylabel('Variation')
plt.xlabel('Number of PC')
plt.show()
```

```
Размерность данных
 (201669, 10)

Вклад каждого фактора в объяснение вариации
 [0.2979766397 0.1496007962 0.1217110055 0.1007219879 0.0999739517
 0.0975640598 0.0735527529 0.055468798  0.0024871325 0.0009428757]

Рост доли объясненной вариации с увеличением числа главных факторов
[0.298  0.4476 0.5693 0.67   0.77   0.8675 0.9411 0.9966 0.9991 1.    ]
```



5. Задание

1. Воспроизведите вычисления, представленные в теоретическом материале практической работы. Подтвердите выводы.

2. Рассмотрите набор данных Turkiye Student Evaluation:
    a) Опишите исследуемые данные
    b) Выберите данные по одному предмету (class) и выполните анализ главных компонент. Выделите главные факторы, дайте интерпретацию (или покажите, что этого сделать нельзя).
    c) Выберите два предмета, которые проводил один и тот же преподаватель. Снова выполните анализ главных компонент, выделите главные факторы, постарайтесь дать интерпретацию. Сравните результаты с предыдущим пунктом.
    d) Выполните PCA для всего набора данных. Также сравните результаты с пунктами выше.
    e) Повторите вычисления из пунктов b - d, но для нестандартизованных данных. Сравните с соответствующими результатами, полученными на стандартизованных данных.

Работа № 3.4 Наивный байесовский классификатор

1. Теоретические сведения

Метод классификации, основанный на наивном байесовском классификаторе, является алгоритмом обучения с учителем, в котором применяется теорема Байеса со строгим (наивным) предположением о независимости между каждыми парами признаков. Предположение о независимости позволяет избавиться от сложной схемы оценки параметров классификатора. Это позволяет применять алгоритм на больших выборках. Также классификация оказывается достаточно точной: недостаточной для высокоточных систем классификации, однако удовлетворительной для грубой оценки и сравнения с другими алгоритмами.

Вероятностная модель для классификатора – это условная модель $P(y \mid x_1, \dots, x_n)$ над независимой переменной класса y и признаками $x_1, \dots, x_n$ по теореме Байеса

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

и предположении независимости, получаем:

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y),$$

Для всехi, это соотношение упрощается

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Поскольку $P(x_1, \dots, x_n)$ является постоянной величиной с учетом входных данных, мы можем использовать следующее правило классификации:

$$P(y \mid x_1, \dots, x_n) \propto P(y)\prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = arg\max_{y} P(y)\prod_{i=1}^{n} P(x_i \mid y),$$

и использовать оценку апостериорного максимума для оценки $P(y)$ и $P(x_i \mid y)$. Различные наивные байесовские классификаторы различаются, в основном, допущениями, которые они делают относительно $P(x_i \mid y)$.

Причина, по которой наивные байесовские модели столь эффективны, заключается в том, что они оценивают параметры, рассматривая каждый признак отдельно и по каждому признаку собирают простые статистики классов.

Положительные и отрицательные стороны наивного байесовского алгоритма

Положительные стороны:

Классификация, в том числе многоклассовая, выполняется легко и быстро.

Когда допущение о независимости выполняется, НБА превосходит другие алгоритмы, такие как логистическая регрессия (logistic regression), и при этом требует меньший объем обучающих данных.

НБА лучше работает с категорийными признаками, чем с непрерывными. Для непрерывных признаков предполагается нормальное распределение, что является достаточно сильным допущением.

Отрицательные стороны:

Если в тестовом наборе данных присутствует некоторое значение категорийного признака, которое не встречалось в обучающем наборе данных, тогда модель присвоит нулевую вероятность этому значению и не сможет сделать прогноз. Это явление известно под названием «нулевая частота» (zerofrequency). Данную проблему можно решить с помощью сглаживания. Одним из самых простых методов является сглаживание по Лапласу (Laplacesmoothing).

Хотя НБА является хорошим классификатором, значения спрогнозированных вероятностей не всегда являются достаточно точными. Поэтому не следует слишком полагаться на результаты, возвращенные методом *predict_proba*.

Еще одним ограничением НБА является допущение о независимости признаков. В реальности наборы полностью независимых признаков встречаются крайне редко.

Приложения наивного байесовского алгоритма

Классификация в режиме реального времени. НБА очень быстро обучается, поэтому его можно использовать для обработки данных в режиме реального времени.

Многоклассовая классификация. НБА обеспечивает возможность многоклассовой классификации. Это позволяет прогнозировать вероятности для множества значений целевой переменной.

Классификация текстов, фильтрация спама, анализ тональности текста. При решении задач, связанных с классификацией текстов, НБА превосходит многие другие алгоритмы. Благодаря этому, данный алгоритм находит широкое применение в области фильтрации спама (идентификация спама в электронных письмах) и анализа тональности текста (анализ социальных медиа, идентификация позитивных и негативных мнений клиентов).

Рекомендательные системы. Наивный байесовский классификатор в сочетании с коллаборативной фильтрацией[2] (collaborative filtering) позволяет реализовать рекомендательную систему. В рамках такой системы с помощью методов машинного обучения и интеллектуального анализа данных новая для пользователя информация отфильтровывается на основании спрогнозированного мнения этого пользователя о ней.

2. Задача: Рубрикация новостных статей

Имеется коллекция новостных статей

$$D = \{d_1, d_2, \ldots, d_n\}$$

Имеется множество рубрик новостей

$$Y = \{y_1, y_2, \ldots, y_k\}$$

Необходимо построить модель, которая для заданной статьи определяет, к какой рубрики она относится

Векторное представление текстов

Словарь W – множество слов (после предобработки, нормализации, удаления стоп-слов), $|W| = m$

Документы D – множество статей, $|D| = n$

Статья x представляется как вектор

$$x = (x_1, x_2 \ldots, x_m)$$

Document-termmatrix$X$: строки – документы, столбцы – слова

[2]**Коллаборативная фильтрация** (англ. *collaborative filtering*)—это один из методов построения прогнозов (рекомендаций) в рекомендательных системах, использующий известные предпочтения (оценки) группы пользователей для прогнозирования неизвестных предпочтений другого пользователя.

Элементы матрицыx_{dt}

- Бинарные: $\begin{cases}1, t \in d\\0, t \notin d\end{cases}$
- Termfrequency (tf): сколько раз слово t встречается в документеd или производная от этого величина
- TF-IDF: $tfidf(t,d,D) = tf(t,d) \cdot idf(t,D)$,

$$idf(t,D) = log\frac{n}{1+|\{d \in D: t \in d\}|}$$

3. Байесовский классификатор

Задача: необходимо найти наиболее вероятное значение $y \in Y$ класса объекта $x = (x_1, x_2 \ldots, x_m)$ при условии заданных признаков объекта.

$$y(x) = \underset{y \in Y}{argmax} p(y|x)$$

Формула Байеса

апостериорная вероятность y при условии x $p(x|y)$ — *априорная вероятность класса y* $p(y)$

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

априорная вероятность объекта x $p(x)$

Максимум вероятности:

$$y(x) = \underset{y \in Y}{argmax} p(x|y)p(y)$$

Часто по умолчанию предполагают значения классов равновероятными

$$y(x) = \underset{y \in Y}{argmax} p(x|y)$$

Рубрикация новостных статей

$$y(x) = \underset{v \in Y}{argmax} p(x_1 = a_1, x_2 = a_2, \ldots, x_m = a_m|y = v)p(y = v)$$

Оценка $p(y = v)$: частота встречаемости статей рубрики v в коллекции $p(x_1 = a_1, x_2 = a_2, \ldots, x_m = a_m|y = v)$ - вероятность в точности такого набора слов. Оценить невозможно, т.к. нет такой статистики

Наивный Байес

Предположение об условной независимости атрибутов в общем случае неверно. Но эти зависимости совпадают для разных классов и «сокращаются» при оценке вероятностей. Например, грамматические зависимости остаются неизменными для всех рубрик новостей.

Наивный Байес хорошо показывает себя при решении задачи классификации, однако он не всегда может быть пригоден для оценки вероятностей.

Предположим условную независимость атрибутов при условии данного значения целевой функции

$$p(x_1 = a_1, x_2 = a_2, \ldots, x_m = a_m|y = v) =$$
$$= p(x_1 = a_1|y = v)p(x_2 = a_2|y = v) \ldots p(x_m = a_m|y = v)$$

Генеративная модель

Модель, по которой порождается документ:

MultinomialNaïveBayes: | Документ – это последовательность событий. Каждое событие –

BernoulliNaïveBayes:	это случайный выбор одного слова из словаря Документ – это вектор бинарных атрибутов, показывающих, встретилось ли в документе то или иное слово

Классификатор с мультиномиальным распределением (Multinomial Naïve Bayes)

Используется в случае дискретных признаков. Например, в задаче классификации текстов признаки могут показывать, сколько раз каждое слово встречается в данном тексте.

Пусть x_i - частота встречаемости слова *i* в документе

Тогда вероятность слова *i* в документе класса *v* оценивается как

$$\hat{\theta}_{vi} = \frac{N_{vi} + \alpha}{N_v + \alpha m}, где$$

$N_{vi} = \sum_{x:x\in D, y(x)=v} x_i \qquad N_v = \sum_{i=1}^{m} N_{vi} \alpha \geq 0$

Классификация:

$$y(x) = \underset{v\in Y}{argmax} \prod_{i=1}^{m} \hat{\theta}_{vi}^{xi}\, p(y = v)$$

Классификатор с распределением Бернулли (Bernoulli Naïve Bayes)

Используется в случае двоичных дискретных признаков (могут принимать только два значения: 0 и 1). Например, в задаче классификации текстов с применением подхода «мешок слов» (bagofwords) бинарный признак определяет присутствие (1) или отсутствие (0) данного слова в тексте.

Пусть $x_i \in \{0,1\}$ - встречается (1) или нет (0) слово *i* в документе

Тогда вероятность слова *i* в документе класса *v* оценивается как

$$P(x_i|y = v) = \frac{1 + N_{vi}}{2 + |\{x: x \in D, y(x) = v\}|}, где$$

$$N_{vi} = \sum_{x:x\in D, y(x)=v} x_i$$

Классификация:

$$y(x) = \underset{v\in Y}{argmax} p(y = v) \prod_{i=1}^{m} (p(x_i|y = v)a_i + (1 - p(x_i|y = v))(1 - a_i))$$

Особенности применения BernoulliNB и MultinomialNB

BernoulliNB принимает бинарные данные, MultinomialNB принимает счетные или дискретные данные (то есть каждый признак представляет собой подсчет целочисленных значений какой-то характеристики, например, речь может идти о частоте встречаемости слова в предложении). BernoulliNB и MultinomialNB в основном используются для классификации текстовых данных.

MultinomialNB и BernoulliNB имеют один параметр alpha, который контролирует сложность модели. Параметр alpha работает следующим образом: алгоритм добавляет к данным зависящее от alpha определенное количество искусственных наблюдений с положительными значениями для всех признаков. Это приводит к «сглаживанию» статистик. Большее значение alpha означает более высокую степень сглаживания, что приводит к построению менее сложных моделей. Алгоритм относительно устойчив к разным значениям alpha. Это означает, что значение alpha не оказывает значительного влияния на хорошую работу модели. Вместе с тем тонкая настройка этого параметра обычно немного увеличивает правильность.

4. Пример: 20 Newsgroups[3]

URL: http://qwone.com/~jason/20Newsgroups/

- Набор новостных статей «20 Newsgroups»
- 18000 новостных статей из 20 различных рубрик.

РешениевScikit-learn

```
from sklearn.datasetsimport fetch_20newsgroups
from pprintimport pprint
from sklearn.feature_extraction.textimport CountVectorizer
from sklearn.naive_bayesimport MultinomialNB
from sklearn.naive_bayesimport BernoulliNB
from sklearnimport metrics
from sklearn.metricsimport classification_report

# Загрузка данных для обучения и тестирования
newsgroups_train = fetch_20newsgroups(subset='train',
remove=('headers', 'footers', 'quotes'))
newsgroups_test = fetch_20newsgroups(subset='test')
news = fetch_20newsgroups(subset='all')

# Список новостных рубрик
# (совпадает в обучающей  и тестовой выборке)
print("Число наблюдений в обучающей выборке\n",
newsgroups_train.filenames.shape)
print("Число наблюдений в тестовой выборке\n",
newsgroups_test.filenames.shape)
print("Список новостных рубрик\n")
pprint(list(newsgroups_train.target_names))
pprint(list(newsgroups_test.target_names))
# Приведение данных к document-term матрице
vectorizer = CountVectorizer()
sparse_train = vectorizer.fit_transform(newsgroups_train.data)
sparse_test = vectorizer.transform(newsgroups_test.data)
# Размерность данных (в dense и sparse совпадает)
print("Размерность обучающей выборки sparse\n", sparse_train.shape)
print("Размерность тестовой выборки sparse\n", sparse_test.shape)
dense_train = sparse_train#.toarray()
dense_test = sparse_test#.toarray()
print("Размерностьобучающейвыборки dense\n", dense_train.shape)
print("Размерность тестовой выборки dense\n", dense_test.shape)
mnb = MultinomialNB(alpha= 1)
for iin range(0, 100):
# Используйте данные обучения для оценки параметров модели и
прогнозирования
mnb.fit(sparse_train, newsgroups_train.target)
pred = mnb.predict(sparse_test)
# Точностьклассификации
a = metrics.accuracy_score(newsgroups_test.target, pred, normalize=True)
print("Точность классификации – доля верно классифицированных объектов из
тестовой выборки \n", a)
print(classification_report(newsgroups_test.target, pred,
target_names=newsgroups_test.target_names))
clf = BernoulliNB(alpha=1)
for iin range(0, 100):
clf.fit(dense_test, newsgroups_test.target)
pred = clf.predict(dense_test)
```

[3] Данные «The 20 Newsgroups» — это коллекция примерно из 20000 новостных документов, разделенная (приблизительно) равномерно между 20 различными категориями. Коллекция «The 20 newsgroups» стала популярным набором данных для экспериментов с техниками машинного обучения для текстовых приложений, таких как классификация текста или его кластеризация.

```
# Точностьклассификации
a = metrics.accuracy_score(newsgroups_test.target, pred, normalize=True)
print("Точность классификации – доля верно классифицированных объектов из
тестовой выборки \n", a)
print(classification_report(newsgroups_test.target, pred,
target_names=newsgroups_test.target_names))
```

Результат работы программы:

```
Число наблюдений в обучающей выборке
 (11314,)
Число наблюдений в тестовой выборке
 (7532,)
Список новостных рубрик

['alt.atheism',
'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
Размерность обучающей выборки sparse
 (11314, 101631)
Размерность тестовой выборки sparse
 (7532, 101631)
Размерность обучающей выборки dense
 (11314, 101631)
Размерность тестовой выборки dense
 (7532, 101631)
```

```
Точность классификации - доля верно классифицированных объектов из тестовой выборки
0.6355549654806161
                          precision    recall  f1-score   support
alt.atheism       0.84      0.31      0.46       319
comp.graphics       0.54      0.72      0.62       389
 comp.os.ms-windows.misc       0.20      0.00      0.01       394
comp.sys.ibm.pc.hardware       0.59      0.62      0.60       392
comp.sys.mac.hardware       0.97      0.38      0.55       385
comp.windows.x       0.39      0.89      0.54       395
misc.forsale       0.93      0.59      0.72       390
rec.autos       0.93      0.62      0.74       396
rec.motorcycles       1.00      0.63      0.77       398
rec.sport.baseball       1.00      0.63      0.78       397
rec.sport.hockey       0.92      0.91      0.91       399
sci.crypt       0.45      0.95      0.61       396
sci.electronics       0.75      0.39      0.52       393
sci.med       0.64      0.84      0.73       396
sci.space       0.73      0.86      0.79       394
soc.religion.christian       0.49      0.93      0.64       398
talk.politics.guns       0.69      0.66      0.68       364
talk.politics.mideast       0.61      0.86      0.72       376
talk.politics.misc       0.51      0.53      0.52       310
talk.religion.misc       0.94      0.06      0.11       251
                accuracy                           0.64      7532
               macro avg       0.70      0.62      0.60      7532
            weighted avg       0.70      0.64      0.61      7532
Точность классификации - доля верно классифицированных объектов из тестовой выборки
0.780801911842804
                          precision    recall  f1-score   support
alt.atheism       0.95      0.46      0.62       319
comp.graphics       0.96      0.48      0.64       389
 comp.os.ms-windows.misc       0.89      0.85      0.87       394
comp.sys.ibm.pc.hardware       0.72      0.94      0.81       392
comp.sys.mac.hardware       0.74      0.91      0.82       385
comp.windows.x       0.94      0.86      0.90       395
misc.forsale       0.37      0.98      0.54       390
rec.autos       0.85      0.91      0.88       396
rec.motorcycles       0.67      0.97      0.80       398
rec.sport.baseball       0.87      0.89      0.88       397
rec.sport.hockey       0.98      0.94      0.96       399
sci.crypt       0.86      0.91      0.88       396
sci.electronics       0.88      0.88      0.88       393
sci.med       0.90      0.74      0.82       396
sci.space       0.94      0.87      0.90       394
soc.religion.christian       0.73      0.82      0.77       398
talk.politics.guns       0.80      0.76      0.78       364
talk.politics.mideast       0.99      0.68      0.80       376
talk.politics.misc       0.99      0.32      0.48       310
talk.religion.misc       1.00      0.02      0.04       251
                accuracy                           0.78      7532
               macro avg       0.85      0.76      0.75      7532
            weighted avg       0.85      0.78      0.77      7532
```

5. Задания

Для набора данных «20 Newsgroups»

1. Подобрать оптимальное значение параметра α из интервала (0, 1)
2. Обучить классификатор с разными априорными вероятностями классов: равными и соответствующими долям классов в обучающей выборке

Работа № 3.5 Метод опорных векторов

1. Теоретические сведения

Метод опорных векторов (МОВ, англ. Support Vector Machine, SVM) - это техника машинного обучения с учителем. Она используется в классификации, может быть применена к регрессионным задачам.

Метод определяет границу принятия решения (далее ГПР) вместе с максимальным зазором, который разделяет почти все точки на два класса, оставляя место для неправильной классификации.

Цель МОВ — определить гиперплоскость (также называется «разделяющей» или ГПР), которая разделяет точки на два класса.

Для ее визуализации представим двумерный набор данных:

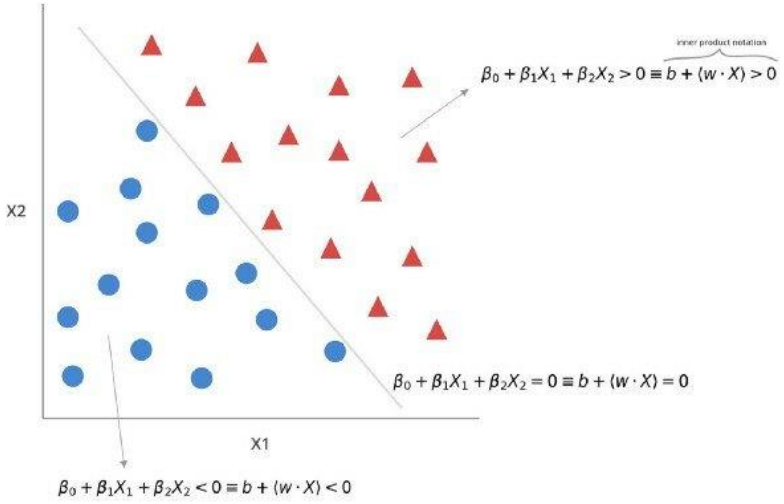


Рисунок. Гиперплоскость, которая полностью разделяет точки на двакласса.

Математическая постановка

Пусть даны два линейно разделимых класса объектов

Мы можем описать все точки разделяющей гиперплоскости используя вектор-нормаль к этой гиперплоскости:

$$\langle w, x \rangle - b = 0$$

Данная разделяющая гиперплоскость находится в «разделяющей полосе», которую мы также можем задать уравнениями:

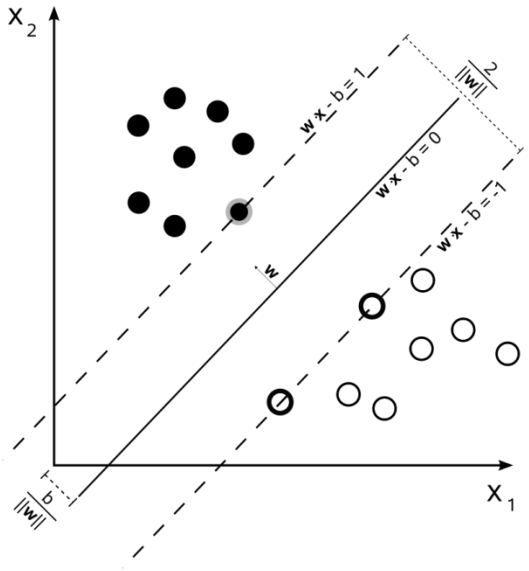
$$\langle w, x \rangle - b = -1 \qquad \langle w, x \rangle - b = 1$$

Можно найти ширину данной полосы как

Для машины опорных векторов необходимо найти разделяющую гиперплоскость, которая задает полосу максимальной ширины. Задача оптимизации:

$$\frac{1}{2}\|w\|^2 \to min$$

$$y_i(\langle w, x_i \rangle - b) \geq 1, \forall i = 1, \dots, n$$



Решение оптимизационной задачи

Для решения оптимизационной задачи с прошлого слайда можно воспользоваться известным методом множителей Лагранжа (Lagrangianrelaxation). Коэффициенты лямбда неотрицательны, поэтому если некоторые точки нарушают ограничения, заданные задачей, значение функции увеличивается. Устремив эту функцию к минимуму по w, b мы будем стремиться к тому, чтобы как можно больше точек не нарушало ограничения. Таким образом, 2-ая часть уравнения будет вносить неположительную часть в функцию и максимальное значение функции будет $\frac{\|w\|^2}{2}$. Максимизируя после этого по неотрицательным лямбда мы получим некоторую нижнюю границу оптимального значения.

$$L_P = \frac{\|w\|^2}{2} - \sum_{i=1}^{n} \lambda_i (y_i(< w, x_i > + b) - 1) \rightarrow \max_{\lambda_i \geq 0} \min_{w,b}$$

Взяв частные производные по w, b и приравняв к нулю, мы избавляемся от минимизации по этим переменным и можем заменить w = sum(y_i*lambda_i*x_i), 0 = sum(y_i*lambda_i). Можно перейти к двойственной проблеме, которая максимизируется по неотрицательным лямбда. Для этого нужно взять производные по переменным минимизации (w, b) и полученные значения подставить в прямую задачу. Получается двойственная задача, которую можно максимизировать только по лямбдам.

$$\frac{\partial L_P}{\partial w} = 0 \quad \rightarrow \quad w = \sum_{i=1}^{n} \lambda_i y_i x_i \, \frac{\partial L_P}{\partial b} = 0 \quad \rightarrow \sum_{i=1}^{n} \lambda_i y_i = 0$$

Для вычисления двойственной функции достаточно знать метки классов, а также скалярное произведение двух точек. Значения w и b можно затем найти после решения задачи для lambda.

Двойственная задача максимизации может быть решена с помощью градиентного спуска, где вектор лямбд итерационно обновляется с помощью вектора частных производных функции L_D.

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j < x_i, x_j > \rightarrow \max_{\lambda_i \geq 0}$$

Случай линейно неразделимой выборки

Формулировка епсилон как функции потерь и ограничений снизу эквивалентна. Ошибка епсилон равна 0 если объект расположен за пределами “разделяющей” полосы.

Ошибка от 0 до 1 говорит о том, что объект расположен внутри полосы, но с правильной стороны от разделяющей гиперплоскости. Ошибка больше 1 говорит о том, что объект классифицирован неверно. Данная формулировка позволяет классифицировать объекты с ошибкой, что обязательно происходит для линейно неразделимой выборки.

Для неразделимых классов вводится функция потерь

$$\varepsilon_i = max\big(0, 1 - y_i(\langle w, x_i\rangle - b)\big)$$

Задача оптимизации:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\varepsilon_i \to min$$

$$y_i(\langle w, x_i\rangle - b) \geq 1 - \varepsilon_i, \quad \varepsilon_i \geq 0, \qquad \forall i = 1, \dots, n$$

Данная форма SVM называется C-classification.

Метод множителей Лагранжа также решает оптимизационную задачу для случая C-classification. Добавляются новые слагаемые с переменной эпсилон и минимизация L_P происходит по трём переменным $-w, b, \varepsilon$. Решение оптимизационной задачи для постановки soft-margin мало отличается от исходного решения оптимизационной задачи, поскольку частные производные по w, b остаются те же самые. Добавляется лишь производная по эпсилон, которая даёт дополнительное ограничение на лямбда – лямбда от 0 до C. Поскольку бета также является множителем Лагранжа и больше или равно нуля, а лямбда также больше нуля, то лямбда не может быть больше C.

Подставив найденные производные в исходную задачу, мы получим ту же самую двойственную проблему, потому что все слагаемые с эпсилон сократятся. Максимизируя двойственную проблему по всем лямбда от 0 до C можно найти решение задачи.

- Метод множителей Лагранжа:

$$L_P = \frac{\|w\|^2}{2} + C\sum_{i=1}^{n}\varepsilon_i - \sum_{i=1}^{n}\lambda_i(y_i(< w, x_i > + b) - 1 + \varepsilon_i) - \sum_{i=1}^{n}\beta_i\varepsilon_i \to \max_{\lambda_i \geq 0}\min_{w,b,\varepsilon_i}$$

- Производные:

$$\frac{\partial L_P}{\partial w} = 0 \;\to\; w = \sum_{i=1}^{n}\lambda_i y_i x_i \qquad \frac{\partial L_P}{\partial b} = 0 \;\to \sum_{i=1}^{n}\lambda_i y_i = 0$$

$$\frac{\partial L_P}{\partial \varepsilon_i} = 0 \;\to\; C - \lambda_i = \beta_i \geq 0, \forall i$$

- Двойственная проблема:

$$L_D = \sum_{i=1}^{n}\lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j < x_i, x_j > \to \max_{0 \leq \lambda_i \leq C}$$

Предсказание

Функция предсказания представляет из себя просто определение расположения точки относительно разделяющей гиперплоскости. W заменена на значение своей производной. В функции предсказания также используется функция ядра (то есть простое скалярное произведение, либо одна из функций, его заменяющих) для внутреннего произведения двух точек. Если какая-то точка не лежит на границе «разделяющей» полосы, то максимальное значение функции будет достигаться если соответствующая лямбда = 0 (см. постановку прямой задачи с множителями лагранжа). Те точки, для которых лямбда не равно нулю и есть опорные, поскольку только от них зависит результат функции предсказания.

По сути, процесс обучения SVM на тренировочной выборке просто представляет из себя процесс решения двойственной задачи оптимизации.

Для предсказания результата алгоритма, используется функция sign:

$$F(z) = sign(\sum_{i=1}^{n} \lambda_i y_i \langle x_i, z \rangle + b)$$

Для $\lambda_i=0$ точка x_i не является «опорной», таким образом, в сумму, которая определяет класс нового объекта, влияние вносят только «опорные» точки.

Использование метода SVM

- Использован набор данных BanknoteAuthentification
- В качестве тестовой выборки взяты 107 последних объектов класса «0» (настоящие банкноты) и 119 объектов класса «1» (фальшивки). Остальные объекты используются в качестве обучающей выборки.
- Исходные параметры алгоритмов SVM взяты одинаковыми для разных библиотек.

Ядра (KernelTrick)

Вместо скалярного произведения точек в двойственной проблеме можно использовать специальные функции, называемые **ядрами**.

Линейное ядро—это аналог применения линейных преобразований к пространству объектов. Предположим, вы увеличиваете исходное пространство объектов возведением во вторую степень. Вы применили квадратичную функцию к исходному набору объектов. Теперь в этом расширенном пространстве есть оригинальная функция и ее квадратичная версия. Здесь неявно существует функция, которая сопоставляет эти два пространственных объекта.

$$x_1, x_2, x_3 \rightarrow x_1, x_1^2, x_2, x_2^2, x_3, x_3^2$$

Расширение пространства объектов с помощью квадратичной версии исходных.

Данные функции переводят пространство, в котором находятся наши точки в пространство большей размерности, что может привести к улучшенной разделимости классов. С полиномиальными ядрами вы проецируете исходное пространство объектов в полиномиальное. Граница, разделяющая классы, определяется полиномом более высокого порядка.

Использование ядер отличает классификаторы от метода опорных векторов, что открывает путь к решению более сложных задач. Но увеличение пространства признаков означает рост вычислительных требований. При большом пространстве функций подгонка модели станет дорогостоящей с точки зрения времени и ресурсов.

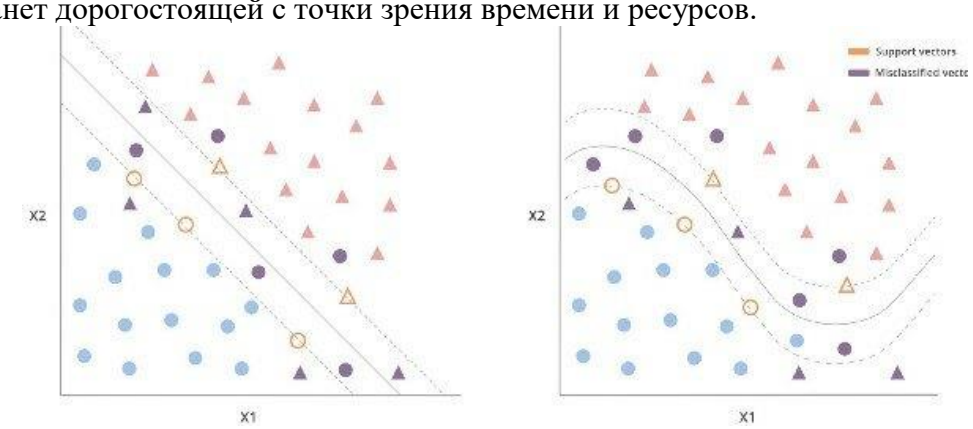


Рисунок. Граница решения и запасдля МОВ, наряду с соответствующими опорными векторами, используют линейное (справа) и полиномиальное ядро(слева).

Плюсы и минусы SVM

- Плюсы:
  - это наиболее быстрый метод нахождения решающих функций;

  - метод сводится к решению задачи квадратичного программирования в выпуклой области, которая всегда имеет единственное решение;
  - метод находит разделяющую полосу максимальной ширины (для заданных параметров), что позволяет в дальнейшем осуществлять более уверенную классификацию (и интерпретацию);
- Минусы
  - метод чувствителен к шумам и стандартизации данных;
  - не существует общего подхода к автоматическому выбору ядра, его параметров и построению спрямляющего подпространства в целом в случае линейной неразделимости классов.

2. Задача – выявление фальшивых банкнот

БазаBanknoteauthentification:
https://archive.ics.uci.edu/ml/datasets/banknote+authentication
Объекты представляют из себя характеристики изображений банкнот
- 1372 объекта
- 4 признака:
  - энтропия изображения
  - коэффициенты дисперсии, ассиметрии и эксцесса вейвлет-преобразования изображения
- Класс (фальшивые или настоящие)

Способ решения – классификация.
Методклассификации – SVM (Support Vector Machine).
Положительные стороны SVM:
  - быстрый метод классификации;
  - метод сводится к решению задачи квадратичного программирования в выпуклой области, которая обычно имеет единственное решение;
  - метод позволяет осуществлять более уверенную классификацию, чем другие линейные методы.

Практическое задание

1. Проанализировать разные результаты для набора данных BanknoteAuthentification, в чём разница базовых настроек алгоритма в разных инструментах?
2. Найти наилучшие параметры для данных BanknoteAuthentification, используя технику кросс-валидации.
3. Возможно ли улучшить точность алгоритма для данных AdultIncome, используя другие параметры (gamma, C, параметры, связанные с SVM)? Подобрать параметры, дающие большую точность или показать, что для большого набора параметров точность улучшить не удаётся.

Работа № 3.6 Нейронные сети

Однослойная нейронная сеть

Нейронная сеть - это алгоритм обучения с учителем, который аппроксимирует функцию $f(\cdot): R^{in} \rightarrow R^{out}$ обучением на наборе данных, где in — количество измерений для ввода и out – размерность выхода.

Искусственный нейрон (ИН) – это простейший аналоговый преобразующий элемент, имитирующий поведение биологического нейрона (рисунок 1).

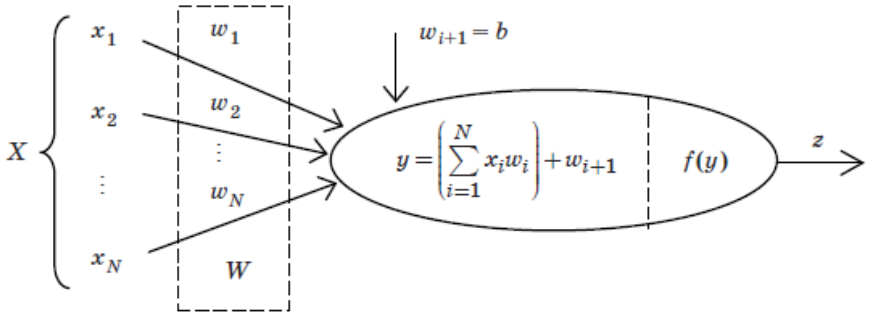


Рисунок 1. – Модель искусственного нейрона

На вход ИН поступает множество (вектор) сигналов. Каждый вход взвешивается – умножается на определенный коэффициент (весовой коэффициент). Сумма всех произведений определяет уровень активации нейрона. Суммирующий блок соответствует соме живого нейрона

Активационная функция F должна быть монотонной. Обычно $F(y)$ принадлежит к интервалу [0,1] или [–1,1]. Чаще используют сигмоидальная активационная функция (2).

Таким образом, ИН выполняет две операции. Сначала вычисляется сумма скалярного произведения вектора весов W и входного вектора X:

$$y = X^T W + b \quad (1)$$

Затем срабатывает активационная сигмоидная функция, определяющая значение выходного сигнала:

$$z = F(y) = \frac{1}{1+\exp(-k \cdot y)}. \quad (2)$$

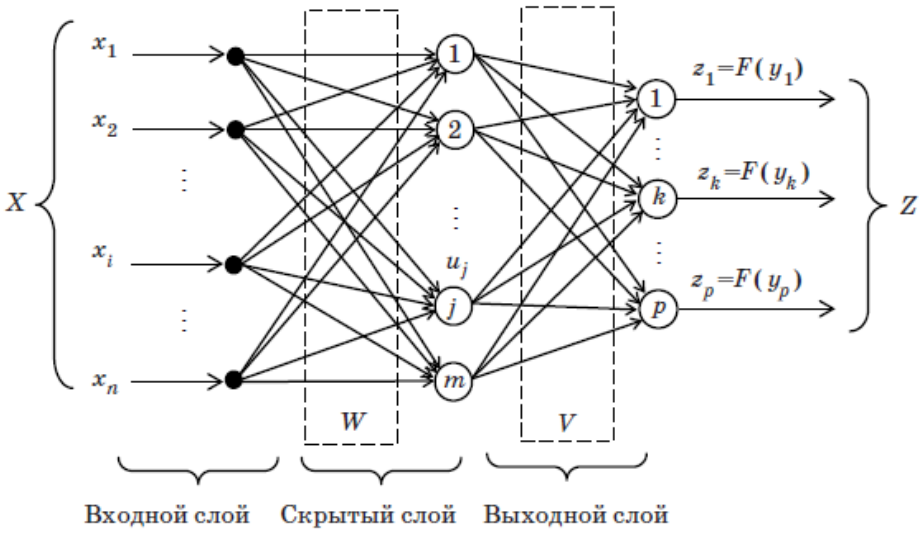


Рисунок 2. – Однослойная НС

Самый левый слой, известный как входной, состоит из набора нейронов $X = x_i|x_1, x_2, \dots, x_n$ представляющие входные функции. Каждый нейрон в скрытом слое преобразует значения из предыдущего слоя с взвешенным линейным суммированием $w_1x_1 + w_2x_2 + \cdots + w_mx_m$, за которой следует нелинейное функциональное преобразование - функция сигмоидного ограничения. Выходной слой получает значения из последнего скрытого слоя и преобразует их в выходные значения.

Нормирование означает приведение каждой компоненты входного вектора к интервалу [0, 1] или [–1, 1]. При известном диапазоне изменения входной переменной [x_{min}, x_{max}] нормирование выполняется по формуле:

$$x_i = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (3)$$

Если значения входной переменной требуется привести к заданному интервалу [a, b], то применяется формула:

$$x_i = \frac{(x - x_{min})/(b - a)}{x_{max} - x_{min}} + a \quad (4)$$

При оценке качества работы ИНС обычно требуется рассматривать среднюю квадратическую ошибку (СКО), определяемую как усредненную на n примерах сумму квадратов разностей между желаемой величиной выхода z_i и реально полученными на сети значениями y_i для каждого i-го примера:

$$J = \frac{1}{N}\sum_{i=1}^{N}(z_m - y_i)^2 \quad (5)$$

Для решения задачи с применением НС, используйте реализации сетей, приложенные к заданию.

Метрики производительности

Для оценки производительности моделей прогнозирования используются следующие метрики:

- Errors = прогнозные значения – действительные значения
- MSE (Mean Squared Error) = $\frac{1}{n}\sum(\text{Прогноз} - \text{Действительныезначения})2$
- RMSE (Root Mean Squared Error) = $\sqrt{MSE}$
- MAE (Mean Absolute Error) = $\frac{1}{n}\sum|\text{Прогноз} - \text{Действительныезначения}|$
- MAPE (Mean Absolute Percentage Error) = $\frac{1}{n}\sum\frac{|\text{Прогноз}-\text{Действительныезначения}|}{\text{Действительныезначения}}$

В случаях, когда большие ошибки важны и они могут привести к серьезным последствиям, следует уделять больше внимания MSE. В том случае, когда приоритетом обладают малые ошибки, следует уделить внимание MAE.

4. Практическое задание

Для данных из работы 3.2 выполнить вычисления по модели нейронной сети прямого распространения и провести анализ полученной модели по СКО.

Качество оценки

	СКО
Линейная регрессия	
Нейронная сеть (конфигурация 1)	
…	
Нейронная сеть (конфигурация n)	

Контрольные вопросы

1. Что такое искусственная нейронная сеть?
2. В чем сходство и отличие линейной регрессии и НС прямого распространения?
3. Какие принципы используются при классификации нейронных сетей?
4. Каково определение искусственного нейрона?
5. Из каких составляющих состоит искусственный нейрон?
6. Какие варианты активационной функции могут быть использованы?
7. Каковы основные парадигмы обучения нейронных сетей?
8. Какие операции могут выполняться при предварительной обработке обучающих данных для нейросети?
9. Как оценить качество обучения нейросети?

Работа № 3.7. Бустинг (AdaBoost, LogitBoost, BrownBoost)

Теоретические сведения

Бустинг — это семейство ансамблевых алгоритмов, суть которых заключается в создании сильного классификатора на основе нескольких слабых. Для этого сначала создаётся одна модель, затем другая модель, которая пытается исправить ошибки в первой. Модели добавляются до тех пор, пока тренировочные данные не будут идеально предсказываться или пока не будет превышено максимальное количество моделей.

AdaBoost был первым действительно успешным алгоритмом бустинга, разработанным для бинарной классификации. Именно с него лучше всего начинать знакомство с бустингом. Современные методы вроде стохастического градиентного бустинга основываются на AdaBoost.

AdaBoost используют вместе с короткими деревьями решений. После создания первого дерева проверяется его эффективность на каждом тренировочном объекте, чтобы понять, сколько внимания должно уделить следующее дерево всем объектам. Тем данным, которые сложно предсказать, даётся больший вес, а тем, которые легко предсказать, — меньший. Модели создаются последовательно одна за другой, и каждая из них обновляет веса для следующего дерева. После построения всех деревьев делаются предсказания для новых данных, и эффективность каждого дерева определяется тем, насколько точным оно было на тренировочных данных.

Так как в этом алгоритме большое внимание уделяется исправлению ошибок моделей, важно, чтобы в данных отсутствовали аномалии.

1. Набор данных Bioresponse

https://www.kaggle.com/c/bioresponse

- 3751объектов, 1777признаков.
- Объект представляет из себя характеристики некоторой молекулы
- Класс объекта – вызвала ли молекула реакцию или нет
- Классификация. Способ классификации - Boosting

Предобработка данных

Пропуски в данных? Нет

Набор данных был разделен на обучающую и тестовую выборку.

Обучающая выборка содержит 3 000 объектов, а тестовая выборка – 751.

Проблема

Можно ли из полученных классификаторов построить агрегированный классификатор с предсказательной точностью выше, чем у найденных классификаторов?

Boosting

Бустинг — это семейство ансамблевых алгоритмов, суть которых заключается в создании сильного классификатора на основе нескольких слабых. Для этого сначала создаётся одна модель, затем другая модель, которая пытается исправить ошибки в первой. Модели добавляются до тех пор, пока тренировочные данные не будут идеально предсказываться или пока не будет превышено максимальное количество моделей.

AdaBoost

AdaBoost был первым действительно успешным алгоритмом бустинга, разработанным для бинарной классификации. Именно с него лучше всего начинать знакомство с бустингом.

Современные методы вроде стохастического градиентного бустинга основываются на AdaBoost.

AdaBoost используют вместе с короткими деревьями решений. После создания первого дерева проверяется его эффективность на каждом тренировочном объекте, чтобы понять, сколько внимания должно уделить следующее дерево всем объектам. Тем данным, которые сложно предсказать, даётся больший вес, а тем, которые легко предсказать, — меньший. Модели создаются последовательно одна за другой, и каждая из них обновляет веса для следующего дерева. После построения всех деревьев делаются предсказания для новых данных, и эффективность каждого дерева определяется тем, насколько точным оно было на тренировочных данных.

Так как в этом алгоритме большое внимание уделяется исправлению ошибок моделей, важно, чтобы в данных отсутствовали аномалии.

Алгоритм AdaBoost строит «сильный» классификатор вида:

$$F_T(x) = sign(f_T(x)) = sign(\sum_{t=1}^{T} \beta_t h(x, a_t))$$

где$w_t \in \mathbb{R}$, $h(x, a_t): X \times A \to \{-1,1\}$ - «слабый» классификатор, принадлежащий некоторому семейству классификаторов H, а A – пространство параметров этого семейства.

Пример H - одноуровневые деревья принятия решений.

Для нахождения классификатора $F_T(x)$ алгоритм AdaBoost последовательно ищет оптимальные параметры β_t и a_t $(1 \le t \le T)$ для построения классификатора $F_t(x)$, используя найденный ранее классификатор $F_{t-1}(x)$:

$F_t(x) = sign(f_{t-1}(x) + \beta_t h(x, a_t)), 1 \le t \le T$

при условии $f_0(x) = 0$.

Шаг 1: Инициализируем веса объектов:

$$w_i = \frac{1}{n}, i = 1, \dots, n$$

Шаг 2: Последовательное построение классификаторов $F_t(x), 1 \le t \le T$

Для *t = 1,...,T:*

a) Обучить слабый классификатор $h(x, a_t) \in H$, используя в качества функции потерь:

$$L(a) = \sum_{i=1}^{n} w_i \, I[\![h(x_i, a) \neq y_i]\!]$$

где $I[\![h(x_i, a) \neq y_i]\!]$ - индикатор ошибки.

b) Подсчет β_t:

$$\beta_t = \frac{1}{2} ln \frac{1 - L(a_t)}{L(a_t)}$$

c) Обновление весов объектов:

$$w_i = \frac{w_i e^{-\beta_t y_i h(x_i, a_t)}}{Z_t}, i = 1, \dots, n$$

где $Z_t -$ нормировочный множитель.

Шаг 3: Построение итогового классификатора:

$$F_T(x) = sign(\sum_{t=1}^{T} \beta_t h(x, a_t))$$

AdaBoost (Scikit-learn)

```
classifier =
ensemble.AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),n_estimato
rs=10)
```

```
#обучениеклассификатора
classifier.fit(train[:,0:n],train[:,n])
#предсказание
prediction = classifier.predict(test[:,0:n])
tab = pd.crosstab(index = prediction, columns= test[:,n])

print(tab)
```

LogitBoost

- Основное отличие LogitBoost от AdaBoost состоит в том, что AdaBoost использует экспоненциальную функцию потерь, а LogitBoost – логистическую.
- За счет этого, в некоторых случаях LogitBoost может превосходить по точности AdaBoost, а также быть более устойчивым к шумам в данных.

$$y_i \in \{0,1\}$$

Шаг 1: Инициализируем переменные и вероятность того, что объект относится к классу 1:

$$f_T(x) = 0$$

$$w_i = \frac{1}{n} \text{ и } p(x_i) = \frac{1}{2}, i = 1, \dots, n$$

Шаг 2: Последовательное построение классификаторов $F_t(x), 1 \le t \le T$

Для *t = 1,...,T:*

a) Расчет w_i и z_i:

$$w_i = p(x_i)(1 - p(x_i))$$

$$z_i = \frac{y_i - p(x_i)}{p(x_i)(1 - p(x_i))}$$

b) Обучить $h(x, a_t) \in H$, используя в качества функции потерь:

$$L(a) = \sum_{i=1}^{n} w_i \, (h(x_i, a) - z_i)^2$$

c) Обновление $f_T(x)$ и $p(x)$:

$$f_T(x) = f_T(x) + \frac{1}{2} h(x, a_t) p(x) = (e^{f_T(x)})/(e^{f_T(x)} + e^{-f_T(x)})$$

Шаг 3: Построение итогового классификатора:

$$F_T(x) = \begin{cases} 1, \text{ если } sign(f_T(x)) \ge 0 \\ 0, \text{если } sign\big(f_T(x)\big) < 0 \end{cases}$$

BrownBoost

- В алгоритме BrownBoost используется дополнительная переменная – «время» работы алгоритма.
- Алгоритм BrownBoost более устойчив к шумам в данных (McDonald R. A. et al. An empirical comparison of three boosting algorithms on real data sets with artificial class noise).

$$y_i \in \{-1,1\}$$

Шаг 1: Подсчитываем «время» работы алгоритма c:

$$c = erfinv^2(1 - \varepsilon)$$

где ε – заданная точность классификации для функции потерь $\frac{1}{n}\sum_{i=1}^{n} |F_T(x_i) - y_i|$,

$erfinv$ – обратная функция к функции $erf(z) = \frac{2}{\pi}\int_0^z e^{-x^2} dx$

Шаг 2: Инициализируем «оставшиеся время» работы алгоритма s_1 и предсказанное значение $r_1(i)$ для объекта i:

$s_1 = c$ и $r_1(i) = 0, i = 1, \dots, n$

Шаг 3: Последовательное построение классификаторов $F_k(x)$

Для *k = 1,2,...* пока $s_k > 0$

a) Задание весов объектов:

$$w_i = \frac{e^{-(r_k(i)+s_k)^2/c}}{Z_k}, i = 1, \dots, n$$

где $Z_k -$ нормировочный множитель.

b) Нахождение слабого классификатора $h(x, a_k) \in H$ такого, что $\sum_{i=1}^{n} w_i h(x_i, a_k) y_i > 0$.

c) Для нахождения $t_k = t^* > 0$, $\beta_k = \beta^*$ таких, что $\gamma^* \leq \nu$ (ν – малая заданная константа, используемая для исключения вырожденных случаев) или $t^* = s_k$ решить дифференциальное уравнение с вещественными переменными γ, β, t:

$$\frac{dt}{d\beta} = \gamma = \frac{\sum_{i=1}^{n} \exp(-\frac{1}{c}(r_k(i) + \beta h(x_i, a_k) y_i + s_k - t)^2)\, h(x_i, a_k) y_i}{\sum_{i=1}^{n} \exp(-\frac{1}{c}(r_k(i) + \beta h(x_i, a_k) y_i + s_k - t)^2)}$$

с краевыми условиями $t = 0, \beta = 0$.

d) Задание s_{k+1} и $r_{k+1}(i)$:

$$r_{k+1}(i) = r_k(i) \beta_k h(x_i, a_k) y_i,\ i = 1, \dots, n$$

$$s_{k+1} = s_k - t_k$$

Шаг 4: Построение итогового классификатора:

$$F_T(x) = sign(\sum_{t=1}^{k-1} \beta_t h(x, a_t))$$

Практическое задание

1. Постройте график зависимости точности классификации и времени работы AdaBoost, LogitBoost, BrownBoost от максимального количества итераций.
2. Постройте график зависимости точности классификации и времени работы BrownBoost от значений параметров ε и ν.
3. Выполните предобработку данных:
   а) анализ и удаление выбросов
   б) снижение размерности
4. Оцените точность классификаторов, найденных AdaBoost, LogitBoost, BrownBoost.

Работа № 3.8. Кластеризация: алгоритмы K-means и EM

1. Теоретические сведения. Метод K-means

Был изобретён в 1950-х годах математиком Гуго Штейнгаузом и почти одновременно Стюартом Ллойдом. Особую популярность приобрёл после работы Маккуина.

- Изобретён в 1950-х годах
- Целевая функция - минимум суммы квадратов расстояний от точек до центров соответствующих им кластеров
- Смешанная (дискретно-непрерывная) задача оптимизации
- K-means – **эвристика**!
- Итерация алгоритма состоит из 2-х этапов
- Количество кластеров задаётся заранее

Описание алгоритма K-means

- **Инициализация**: алгоритм инициализируется центрами кластеров:
  - Первые k точек
  - Случайные k точек
  - Заранее определенные k точек
  - K-means++
  - Другиеалгоритмы (Random Partitioning, Build Algorithm…)
- **Шаг назначения**: известны центры кластеров. Распределение точек по ближайшим кластерам.
- **Шаг обновления**: пересчёт центров кластеров.

При инициализации точки как правило выбираются из набора данных. В некоторых случаях, точки могут быть заданы и не из набора данных, а как некоторые заранее определенные значения.

Если расстояние эвклидово – то среднее

Если Манхеттена – то медиана

K-means++ - каждый следующий центроид выбирается по вероятности – чем дальше расположена точка от текущих известных центров, тем выше вероятность ее выбора.

Назначение:

$$S_i^{t+1} = \{x_p : (x_p - \mu_i^t)^2 \leq (x_p - \mu_j^t)^2, \forall j \neq i\}$$

S_i– кластер с номером i, x– точки для кластеризации, μ– центры кластеров, t– номер шага.

Обновление средних:

$$\mu_i^{t+1} = \frac{1}{|S_i^{t+1}|} \sum_{x_j \in S_i^{t+1}} x_j$$

Оба шага таким образом уменьшают сумму квадратов расстояний.

Действительно, после шага 1 суммарное расстояние уменьшиться поскольку точки могли быть распределены в более дальние кластеры, таким образом их вклад может только уменьшиться. На втором шаге находится среднее значение всех точек, входящих в новый кластер, известно, что именно среднее значение даёт минимальную сумму квадратов расстояний от него до всех значений точек (минимум дисперсии достигается в мат. ожидании).

Алгоритм останавливается после определенного количества итераций либо по достижении сходимости (центры и распределение точек по кластерам не изменилось за итерацию)

K-means

Как правило нужно запускать алгоритм с разным k чтобы найти оптимальное разбиение

Например, вместо нахождения 3 кластеров (1 большой, 2 поменьше, но все явно отделены друг от друга), алгоритм может разбить большой кластер на 2 поменьше, а в третий поместить 2 маленьких.

Для улучшения результатов можно ограниченное число раз перезапустить алгоритм для других начальных центроидов

- Особенности метода:
  - Количество кластеров необходимо определять самостоятельно
  - Теоретически обеспечена сходимость к локальному минимуму
  - На практике локальные минимумы могут не давать логичный результат для кластеризации
  - Результат зависит от выбора начальных центроидов, которых может быть бесконечно много.
  - Старается создавать кластеры примерно одного размера / разброса, выделяет эллиптические(шарообразные) кластеры

1. Задача

- Набор данных –Quake (землетрясения)
- Задача – определить точки сейсмической активности
- Способ решения – кластеризация с выделением центров кластеров
- Методы кластеризации – K-means, EM-алгоритм

http://sci2s.ugr.es/keel/category.php?cat=uns

Набор данных Quakes (землетрясения)

- Объекты – информация о землетрясения
- 2178 объектов
- 4 признака:
  - Глубина точки гипоцентра землетрясения
  - Широта и долгота точки землетрясения
  - Сила землетрясения по шкале Рихтера
- Какую информацию можно найти в этой базе с помощью кластеризации?

```
from sklearnimport cluster
model = cluster.KMeans(n_clusters=n,init='random', algorithm='full',
max_iter=10000)
#Заданиепараметров
clusterobj = model.fit(dataset)
#Нахождениекластеров
print(clusterobj.cluster_centers_)
print(clusterobj.inertia_)
#Распечатка центров полученных кластеров и целевой функции
```

EM-алгоритм

Имя алгоритму было дано в 1977 году в статье авторов Arthur Dempster, Nan Laird, Donald Rubin. До этого метод использовался различными учёными для разных задач без какого-то определения алгоритма. Однако именно эта статья в *Journal of the Royal Statistical Society* дала жизнь EM-методу как инструменту статистического анализа.

- Данные представляются как выборка из смеси распределений (например, нормальных)
- Целевая функция – функция правдоподобия для предложенного набора данных и заданного количества компонент смеси.
- Итерация алгоритма состоит из 2-х этапов
- Количество компонент задаётся заранее

Описание EM-алгоритма

Далее предполагается что распределения нормальные (гауссовы). Под параметрами тогда понимаются вектор средних и матрица ковариаций каждой компоненты смеси, а также априорные вероятности того, из какой компоненты получены данные. В качестве скрытых переменных могут быть взяты переменные z_{ij}, которые представляют из себя апостериорные вероятности получения объекта I из компоненты j. При инициализации, их берут такими, что если объект i получен из компоненты j, то $z_{ij} = 1$, иначе = 0.

Тогда, на шаге E данные апостериорные вероятности могут быть получены через формулу Байеса –f это функции плотности, альфа – априорные вероятности.

На шаге M параметры пересчитываются с помощью весов W довольно очевидным образом.

- Предполагается, что данные X получены из смеси распределений с параметрами ϑ, также предполагается наличие скрытых переменных Z. Параметры перед началом алгоритма инициализируются.
- **Estimation (E-step):** На основе данных и известных параметров вычисляется матрица Z весов или апостериорных вероятностей:

$$w_{ij} = \frac{f\left(x_i \middle| z_{ij}, \theta_j^{s-1}\right) * \alpha_j^{s-1}}{\sum_{r=1}^{k} f(x_i | z_{ir}, \theta_r^{s-1}) * \alpha_r^{s-1}}$$

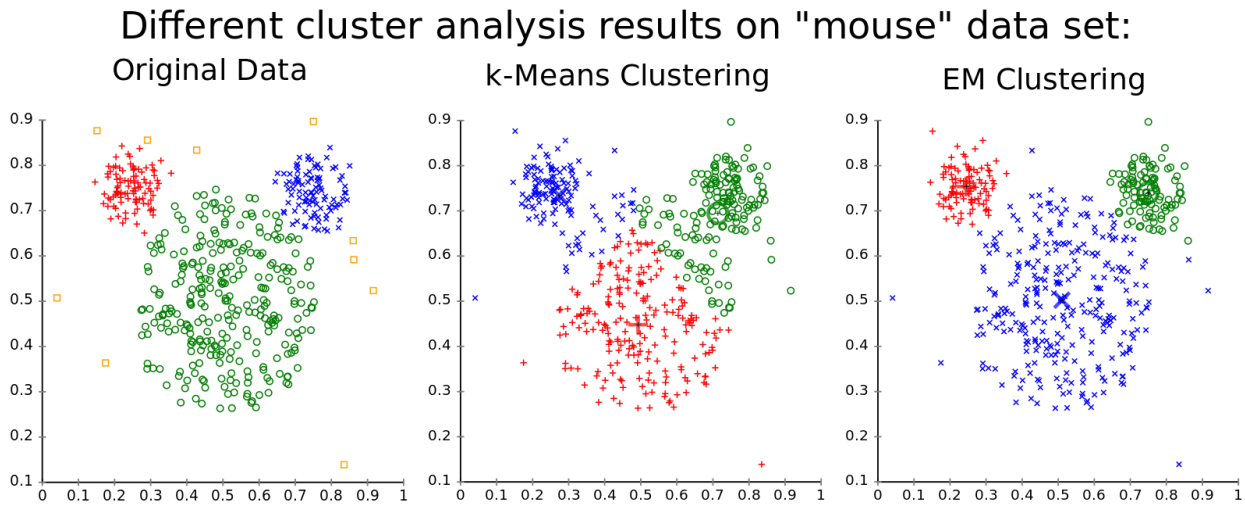
- **Maximization (M-step):** Пересчитываются параметры (средние значения, априорные вероятности, матрицы ковариации)

Особенности EM - алгоритма

Для большей устойчивости алгоритм можно несколько раз перезапускать с разными начальными условиями.

- Количество компонент необходимо определять самостоятельно
- Вектор скрытых переменных вводится таким образом, чтобы:
  - Его было легко найти при известных параметрах
  - Поиск максимума правдоподобия упрощается если известен вектор скрытых переменных
- Теоретически обеспечена сходимость к локальному минимуму
- Локальный минимум сильно зависит от начальной инициализации параметров (неустойчивость по начальным данным)

Картинка внизу хорошо подходит для K-means. Сверху можно увидеть пример не самого хорошего набора данных для K-means кластеризации – видно, что кластеры пытаются быть примерно одного размера



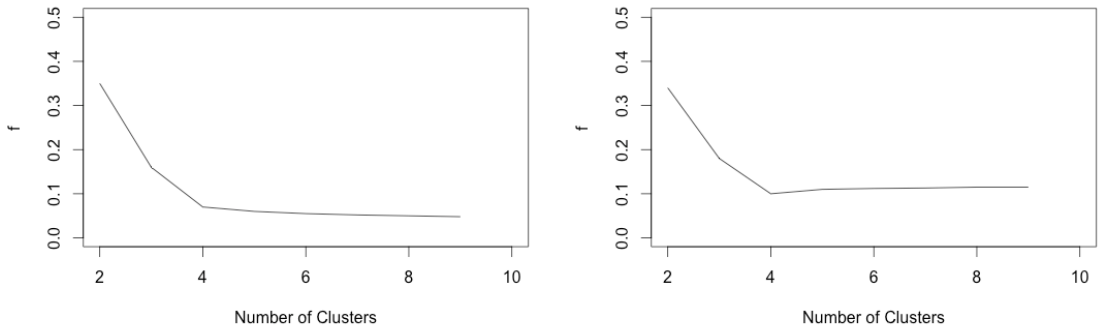
Алгоритмы определения количества кластеров

Вместо суммы внутрикластнерных расстояний можно использовать отношение среднего внутрикластерного расстояния к среднему межкластерному расстоянию.

Индексы как правило обладают некоторым набором значений, при котором можно сделать вывод о высокой степени кластеризации. Наличие соответствующих значений для некоторого выбранного количества кластеров может являться причиной оконгчательного выбора данного числа кластеров.

- Каменистая осыпь: анализируя график суммы внутрикластерных расстояний, эмпирически находится место, где увеличение количества кластеров перестаёт сильно влиять на изменение этой суммы.
- Различныеиндексы– Davies-Bouldin index, Dunn index, Silhouette coefficient.
- Другие эмпирические наблюдения (по количеству объектов в кластерах, по визуальным данным и т.д.).

Слева пример графика функции внутрикластерных расстояний, справа – отношения внутрикластерных расстояний к межкластерным. Можно заметить, что в районе 4 кластеров оба графика «останавливаются». Это может служить эмпирической причиной выбора 4 кластеров в этом конкретном случае




```
from sklearnimport mixture
model = mixture.GaussianMixture(n_components=50, max_iter=10000)
#Задание параметров
model.fit(dataset)
#Нахождение кластеров
print(model.means_)
#Распечатка центров полученных кластеров
```

Практическое задание

1) Построить графики показателей (сумма квадратов расстояний, отношение среднего внутрикластерного расстояния к внекластерному) для различного количества кластеров для набора данных Quake. Определить оптимальное число кластеров, анализируя эмпирическую информацию распределения «очагов».

Как правило нужно запускать алгоритм с разным k чтобы найти оптимальное разбиение

Например, вместо нахождения 3 кластеров (1 большой, 2 поменьше, но все явно отделены друг от друга), алгоритм может разбить большой кластер на 2 поменьше, а в третий поместить 2 маленьких.

Для улучшения результатов можно ограниченное число раз перезапустить алгоритм для других начальных центроидов

Задания для самостоятельной работы

Самостоятельная работа №1. Изучение типовых алгоритмов мягких вычислений в анализе Больших Данных.

Требования:

1. Свободно ориентироваться в алгоритмах мягких вычислений (нейронные сети, нечеткая логика, генетические алгоритмы и др.).
2. Уметь объяснить алгоритмы мягких вычислений при работе с Большими Данными.

Самостоятельная работа №2. Роевые алгоритмы (муравьиные алгоритмы)

Требования:

Проработать контрольные вопросы:

1. Опишите суть алгоритма муравья.
2. Какие классы задач можно решать с помощью алгоритма муравья?
3. Назовите основные этапы алгоритма муравья.
4. Как формируется начальная популяция?
5. Как описывается уравнение движения муравьев?
6. Как вычислить количество фермента, оставленного на каждой грани пути?
7. Как описывается постепенное испарение фермента?
8. Как отобрать наилучшее решение?
9. Приведите примеры решения практических задач, используя алгоритм муравья.

ПРИЛОЖЕНИЕ А. Наборы данных

1. Kaggle

https://www.kaggle.com/

Kaggle ежедневно обновляется энтузиастами и содержит одну из крупнейших библиотек баз данных в интернете.

Kaggle — это платформа машинного обучения, управляемая сообществом. Она содержит множество учебных пособий, которые охватывают сотни реальных проблем МО. Конечно, качество данных может различаться, однако все они совершенно бесплатны. Также есть возможность загрузить в библиотеку свою собственную базу данных.

2. Dataset Search от Google

https://datasetsearch.research.google.com/

DatasetSearch — это надежный источник информации для исследований. В нем все наборы данных отсортированы по:

- актуальности;
- формату файла;
- типу лицензии;
- теме;
- последнему обновлению.

3. Открытые наборы данных Microsoft Azure

https://docs.microsoft.com/en-us/azure/azure-sql/public-data-sets

Открытые наборы данных Azure регулярно обновляются и доступны для разработчиков приложений и исследователей. Они содержат правительственные данные США, другие статистические и научные данные, а также информацию из онлайн-сервисов, которую Microsoft собирает о своих пользователях.

Кроме того, Azure предлагает пользователям набор инструментов, которые помогают создавать собственные облачные базы данных, переносить рабочие нагрузки SQL в Azure при сохранении полной совместимости с SQL Server и создавать управляемые данными мобильные и веб-приложения.

4. Открытые базы данных на Github

https://github.com/awesomedata/awesome-public-datasets#machinelearning

Базы Github – коллекция наборов данных с открытым исходным кодом, разделенных по отраслям. Некоторые из библиотек, которые вы можете там найти будут упомянуты здесь позже.

5. Наборы данных для глубокого обучения

Глубокое обучение основано на использовании искусственных нейронных сетей, применяемых для решения задач. Вместо того, чтобы писать алгоритм для задачи, программист использует обучение и позволяет машине делать прогнозы самостоятельно.

- Набор данных OpenImages от Google очень разнообразен и содержит сложные образцы с несколькими объектами на изображении. Он содержит ограничительные рамки и сегментацию объектов, а также метки, которые помогут ориентироваться в более чем 9 миллионах изображений.
- VisualData — это агрегатор наборов данных для компьютерного зрения, где вы можете найти медицинские пакеты данных для машинного обучения, пакеты данных

изображений и другие интересные образцы для бизнеса, образования и других видов исследований МО.

- xView — это одно из крупнейших общедоступных хранилищ изображений. Оно содержит изображения со всего мира, состоящие из сложных сцен и аннотированные с помощью ограничительных рамок.
- База данных MNIST представляет собой набор моделей для распознавания рукописных цифр. Он содержит обучающий набор из более чем 60 000 примеров и тестовый — из 10 000. На веб-сайте вы также найдете таблицу, в которой сравнивается эффективность различных типов классификаторов, применяемых к этому набору данных. Даже новичок может использовать MNIST для обучения своей модели глубокого обучения.
- CIFAR-10 представляет собой сборник изображений для тренировки алгоритмов компьютерного зрения. Банк данных состоит из 60 000 цветных изображений 32x32 в 10 классах — по 6000 изображений в каждом. Если этого недостаточно, попробуйте набор данных CIFAR-100.

Список использованных источников

1. Теоретический минимум по BigData. Всё, что нужно знать о больших данных. — СПб.: Питер, 2019. — 208 с.: ил. — (Серия «Библиотека программиста»).
2. Радченко И.А, Николаев И.Н. Технологии и инфраструктура BigData. –СПб: Университет ИТМО, 2018. – 52 с..
3. Python для сложных задач: наука о данных и машинное обучение. — СПб.: Питер, 2018. — 576 с.: ил. — (Серия «Бестселлеры O’Reilly»).
4. Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learnи TensorFlow 2, 3-еизд.: Пер. сангл. - СПб.: ООО "Диалектика", 2020. - 848 с.: ил. - Парал. тит. англ.
5. Андреас Мюллер, Сара Гвидо. Введение в Машинное обучение. Руководство для специалистов по работе с данными. Андреас Мюллер, Сара Гвидо. Москва Санкт-Петербург Киев.
6. Python: Искусственный интеллект, большие данные и облачные вычисления. — СПб.: Питер, 2020. — 864 с.: ил. — (Серия «Для профессионалов»).
7. https://scikit-learn.org/stable/index.html [Электронный ресурс]: Документация к scikit-learn (англ.)
8. https://docs.python.org/3/index.html [Электронный ресурс]: Документация к Python (англ.)

Учебное издание

Дмитрий Александрович Оськин

ПРАКТИЧЕСКИЙ АНАЛИЗ ДАННЫХ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON

Рекомендовано методическим советом ДВФУ в качестве учебного пособия для студентов, обучающихся по направлениям подготовки 09.03.03, 09.04.03 «Прикладная информатика (по отраслям)»

Электронная версия оригинал-макета, подготовленного автором.

Усл. печ. л. 7.25. Уч.-изд. л. 5.8. Формат 60×84 1/16
Тираж 100 экз., Заказ №