Министерство науки и высшего образования РФ
ФГБОУ ВО «Ульяновский государственный университет»
Факультет математики, информационных и авиационных технологий

Кафедра телекоммуникационных технологий и сетей

Липатова Светлана Валерьевна

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
для лабораторного практикума
и самостоятельной работы
по дисциплине

«Машинное обучение»

для студентов направления
11.04.02 " Инфокоммуникационные технологии и системы связи"

Ульяновск
2022

Методические рекомендации для семинарских (практических) занятий, лабораторного практикума и самостоятельной работы по дисциплине «Машинное обучение» / составитель: С.В. Липатова - Ульяновск: УлГУ, 2022 –78 с.

Настоящие методические рекомендации предназначены для студентов направления обучения 11.04.02 "Инфокоммуникационные технологии и системы связи". В работе приведены материалы для самостоятельного изучения и выполнения лабораторных работ по темам курса, в том числе примеры кодов на языке Python.

Студентам всех форм обучения следует использовать данные методические рекомендации при подготовке к семинарам, самостоятельной подготовке, а также промежуточной аттестации по дисциплине «Машинное обучение».

Рекомендованы к введению в образовательный процесс

Учёным советом факультета математики, информационных и авиационных технологий УлГУ

протокол № 3/19 от «19» апреля 2022 г.

СОДЕРЖАНИЕ

ОБЩИЕ ВОПРОСЫ

В результате изучения дисциплин «Машинное обучение» студенты должны:

- сформировать системное базовое представление, первичные знания, умения и навыки студентов по основам машинного обучения,
- дать представления о прикладных системах машинного обучения, способах и инструментах их построения,
- дать представление о роли методов и моделей машинного обучения в развитии IT и в научно-техническом прогрессе,
- подготовить студентов к применению концепций машинного обучения при дальнейшем обучении и в решении практических задач.

Методические рекомендации для семинарских (практических) занятий, лабораторного практикума и самостоятельной работы по дисциплине «Машинное обучение» направлены на повышение эффективности освоения знаний, умений, навыков и компетенций, связанных с использованием библиотек на языке Python для решения задач машинного обучения.

Методические рекомендации предлагают указания по всем темам дисциплины «Машинное обучение». Методические рекомендации разбиты по темам и содержат набор вопросов для систематизации теоретического материала, полученного на лекционных занятиях, и самостоятельного изучения теории, вопросы (тесты) для текущего контроля на практических занятиях (семинарах), задачи для усвоения практических навыков. Для лабораторного практикума приведены задания, варианты и рекомендации по выполнению лабораторных работ.

Список литературы и информационного обеспечения, приведённый в конце методических указаний, может служить основой для изучения всех рассматриваемых тем. Дополнительная и учебно-методическая литература могут быть использованы обучающимися для закрепления изучаемого материала.

РЕКОМЕНДАЦИИ ПО ОТДЕЛЬНЫМ ТЕМАМ ДИСЦИПЛИНЫ

Тема 1. Введение в машинное обучение.

Виды обучения по источнику знаний (Рисунок 1):

- Дедуктивное или аналитическое обучение (экспертные системы, источник знаний - эксперт): знания формулируются экспертом и формализованы, имеется механизм вывода.
- Индуктивное обучение (≈статистическое обучение, источник знаний – эмпирический данные): на основе эмпирических данных программа строит общее правило, данные могут быть получены самой программой в предыдущие сеансы ее работы или просто предъявлены ей.
- Комбинированное обучение: включает оба подхода.

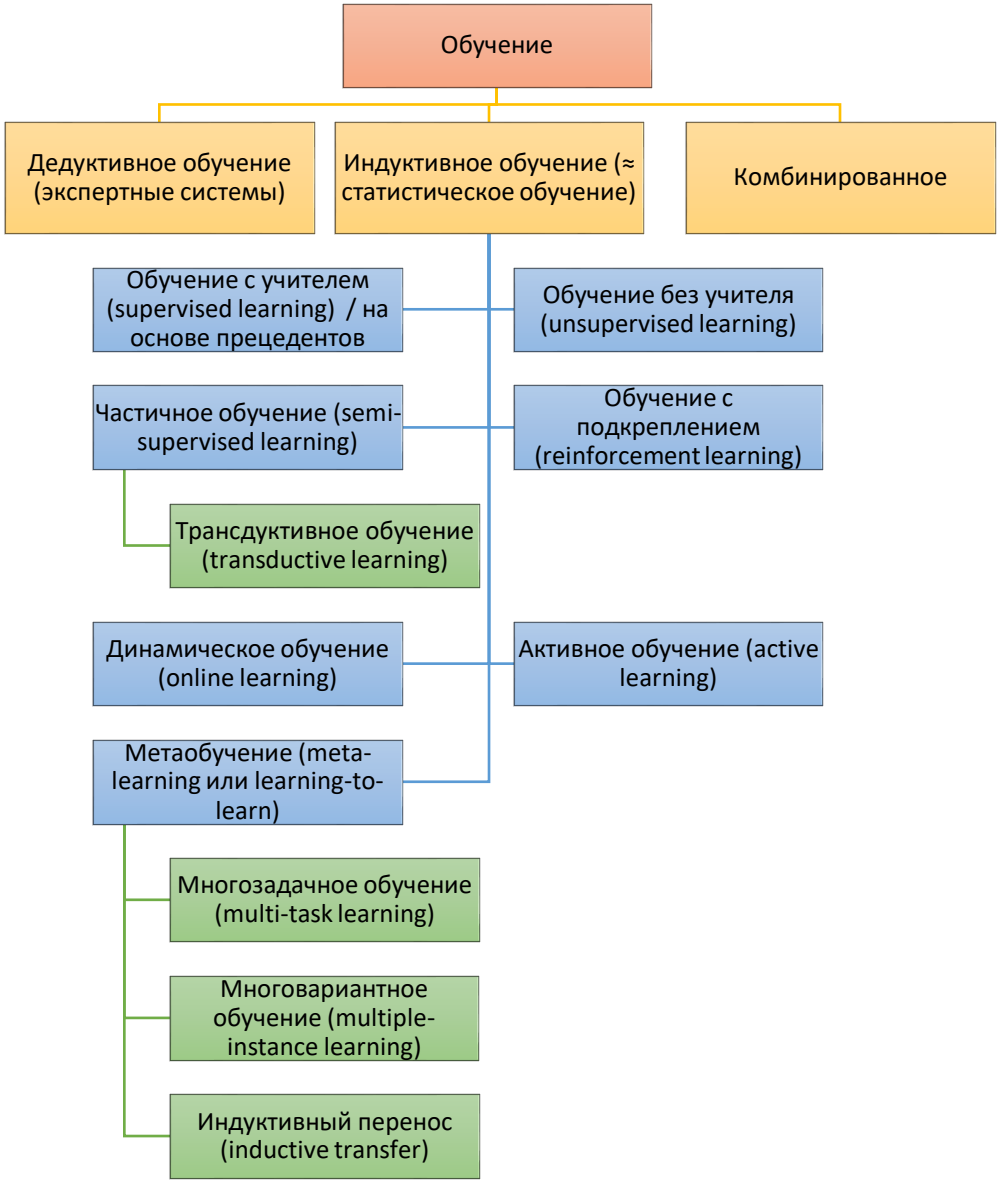


Рисунок 1 – Виды машинного обучения

Ниже представлена примерная схема определения вида обучения для текущей задачи (Рисунок 2) в зависимости от имеющихся данных и вида задачи.

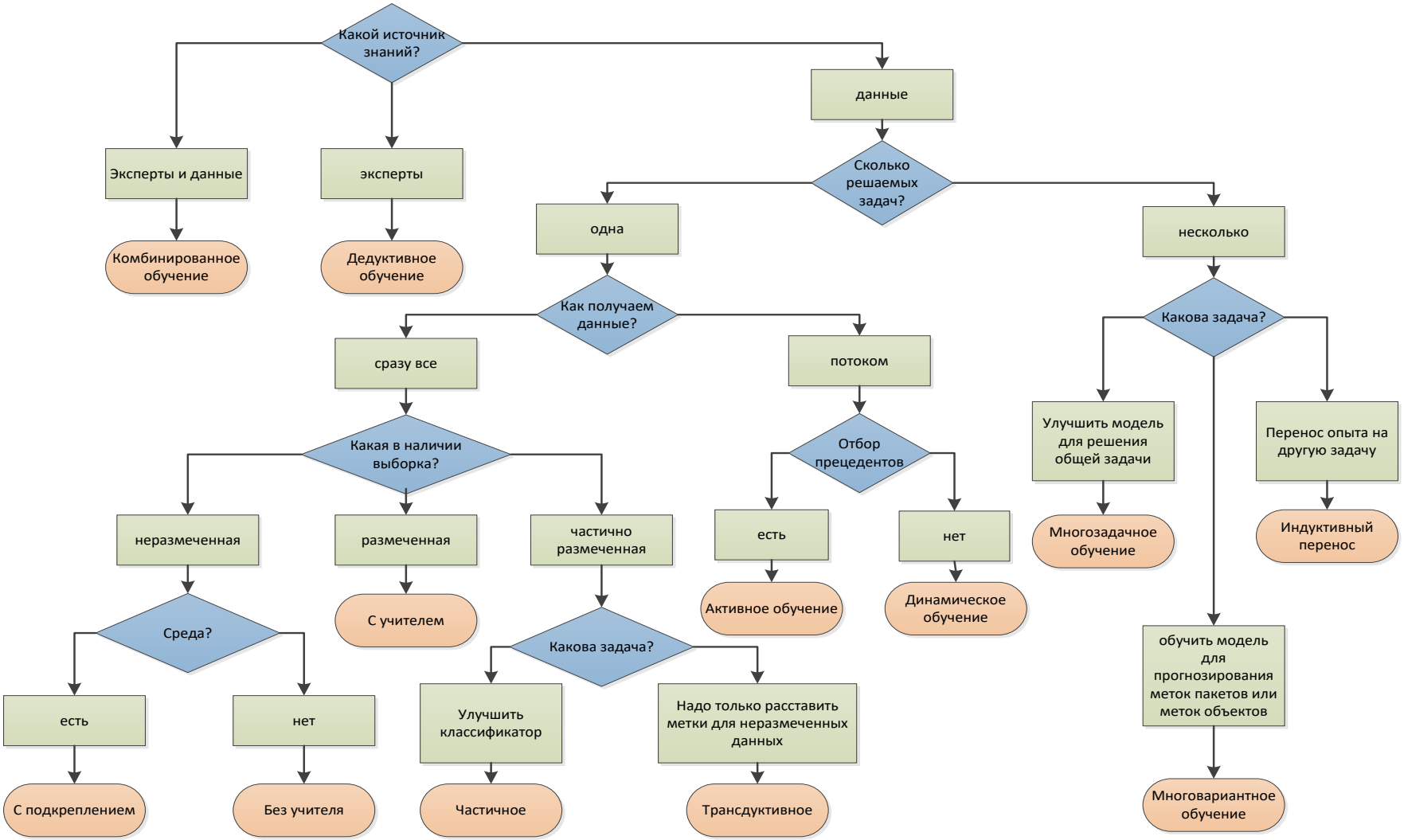


Рисунок 2 – Схема выбора вида обучения для задачи

Обучение с учителем

- **Обучение с учителем (supervised learning) / на основе прецедентов**

Пусть X — множество объектов, Y — множество ответов и имеется некоторая зависимость (детерминированная или вероятностная), позволяющая по $x \in X$ предсказать $y \in Y$. То есть, если эта зависимость детерминированная, то существует некоторая функция $f^*: X \to Y$. Причем изначально, эта зависимость известна только на объектах из обучающей выборки. Задача машинного обучения с учителем — научиться по новым объектам $x \in X$ предсказывать $y \in Y$, т.е восстановить зависимость.[1]

Результат обучения: *индикатор класса, число/вектор.*

Система принудительно обучается с помощью примеров «стимул-реакция» - прецедентов. С точки зрения кибернетики, является одним из видов кибернетического эксперимента. Между входами и эталонными выходами (стимул-реакция) может существовать некоторая зависимость, но она неизвестна. Известна только конечная совокупность прецедентов — пар «стимул-реакция», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость (построить модель отношений стимул-реакция, пригодных для прогнозирования), то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов, так же как и в обучении на примерах, может вводиться *функционал качества*.

Разность между целевым и фактическим выходами модели называется ошибкой обучения (невязкой, остатками), которая минимизируется в процессе обучения и выступает в качестве "учителя".

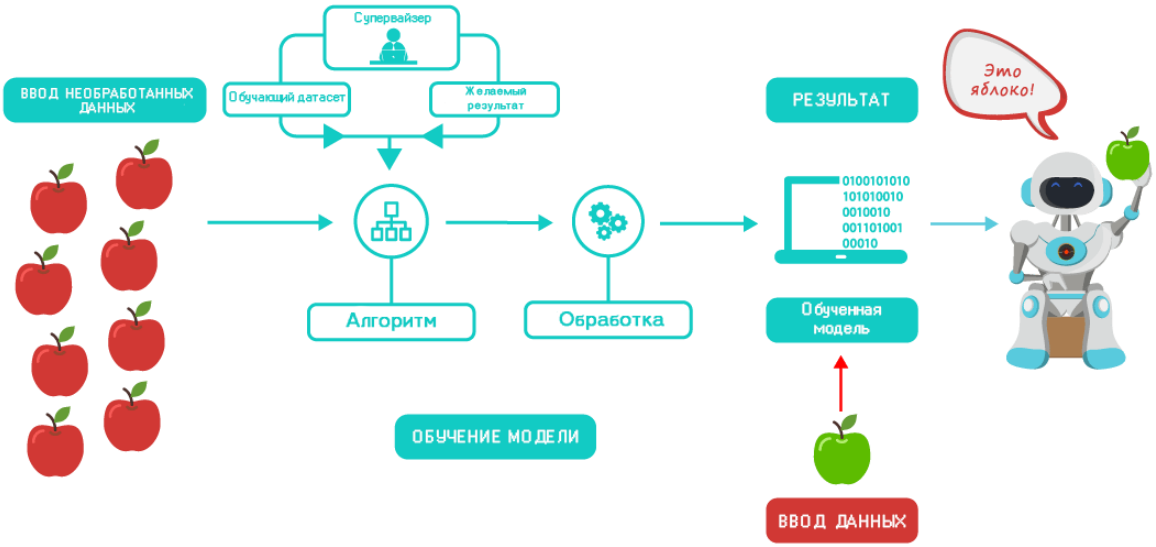


Рисунок 3- Схема работы с учителем (супервайзером)[2]

[1] https://savepearlharbor.com/?p=163675
[2] https://rb.ru/story/ai-dictionary/

Обучение с учителем используется в задачах классификации и регрессии. В первом случае в качестве целевой переменной используется метка класса, а во втором - числовая переменная целого или вещественного типа.

Прецеденты представляют из себя *размеченные данные* — это группа данных с присвоенными справочными тегами или выходной информацией. Например, массив фотографий котов, в котором указано, что это именно фотографии котов. Машина перемалывает такой массив данных и учится угадывать, есть ли кот на новой фотографии, и получает некоторые решающие правила (модель нейронной сети, дерево решений, формулу и т.д.).

Обобщающая способность (качество) решающего правила — это способность решающего правила правильно предсказывать выход для новых объектов, не вошедших в обучающую выборку.

Малое значение функционала качества на обучающей выборке не гарантирует, что построенный алгоритм будет хорошо восстанавливать целевую зависимость на всём пространстве X. Существует опасность переподгонки или переобучения, когда делается попытка описать конкретные данные точнее, чем в принципе позволяет уровень шума в данных и погрешность самой модели.

Переобучение (overfitting) — решающее правило хорошо решает задачу на обучающей выборке, но имеет плохую обобщающую способность.

Недообучение (underfitting) – явление, при котором ошибка обученной модели оказывается слишком большой.

Обучение без учителя

- **Обучение без учителя (unsupervised learning)**

Пусть X — множество объектов. Задача программы — определить, как элементы из X связаны между собой. В частности, решить задачу кластеризации, т.е разбить объекты на группы таким образом, чтобы в одной группе оказались объекты похожие, а в разных — существенно различные.[3]

Результат обучения: *кластеры или правила/деревья решений.*

Методы обучения без учителя используются, когда никаких правильных ответов нет, есть только объекты и их признаки, а задача заключается в том, чтобы определить структуру множества этих объектов.

Алгоритм ищет не пары объект-ответ, а связи между объектами. В случае обучения без учителя, обучающая выборка состоит только из объектов. Алгоритм получает только

[3] https://savepearlharbor.com/?p=163675

сырые входные данные, которые не требуют первичной обработки, анализирует датасет и самостоятельно проводит кластеризацию данных, разделяя их на группы со схожими показателями.

В алгоритмах обучения без учителя выходная ошибка модели на обучающем множестве не вычисляется. Вместо неё используется информация о текущем состоянии параметров модели и примеров обучающего множества.

Основное применение обучения без учителя - построение моделей для кластеризации, поиск ассоциативных правил и аномалий.

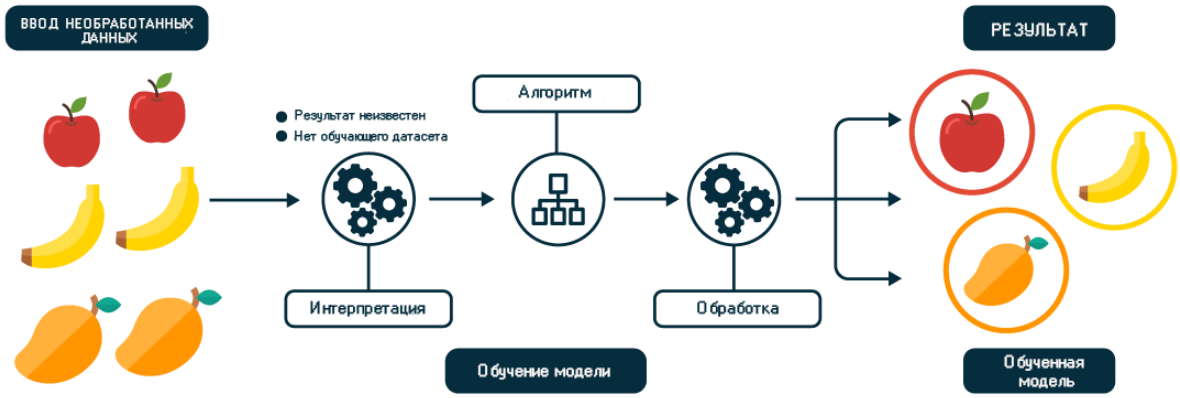


Рисунок 4 – Схема работы без учителя[4]

Если полученный в результате обучения паттерн представляет собой группу, мы называем ее *кластер*. Если паттерн — это правило (например: если вот это, то вот так), мы называем его *ассоциацией*.

Частичное обучение

- **Частичное обучение (semi-supervised learning) / обучение с частичным привлечением учителя / полуавтоматическое обучение**

Пусть $X=\{x_1,\ldots,x_n\}$ *— множество объектов,* Y *— множество ответов и имеется некоторая зависимость (детерминированная или вероятностная), позволяющая по* $x \in X$ *предсказать* $y \in Y$,. $\{x_{n+1},\ldots,x_m\}$ *— множество неразмеченных объектов, принадлежавших* X.

Результат обучения: *индикатор класса, число/вектор.*

При частичном обучении большая часть ответов неизвестна, обычно небольшое количество размеченных данных и большое количество неразмеченных данных (Рисунок 5).

[4] https://rb.ru/story/ai-dictionary/

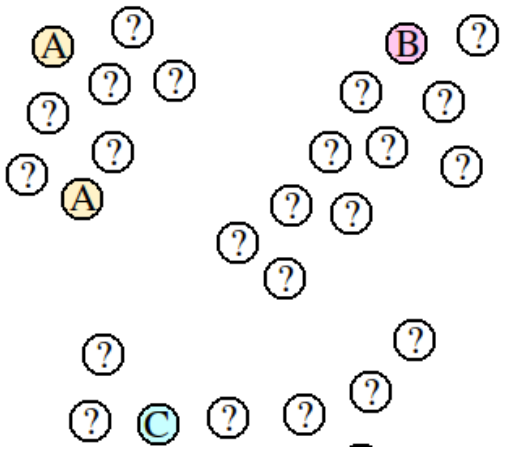


Рисунок 5 – Визуализация входного множества X для задачи частичного обучения[5]

Занимает промежуточную позицию между обучением без учителя (без привлечения каких-либо размеченных данных для тренировки) и обучением с учителем (с привлечением лишь размеченных данных).

Цель полуавтоматической обучения заключается в том, чтобы использовать эту комбинированную информацию для достижения лучших результатов производительности классификации, которую можно получить или путём отбрасывания неразмеченных данных и использование обучения с учителем, или путем отбрасывания меток и использование обучения без учителя.

Полуавтоматическое обучения может принадлежать к трансдуктивного обучения или индуктивного обучения.

У этого метода есть несколько преимуществ. Во-первых, маркировка огромного массива данных — долгий и дорогостоящий процесс. Во-вторых, маркировка всех данных в массиве может привести к появлению в модели систематической ошибки, вызванной человеческим фактором. Включение в модель неразмеченных данных одновременно снижает стоимость обучения алгоритма и позволяет сделать модель более точной.

Трансдуктивное обучение

- **Трансдуктивное обучение (transductive learning)**

Пусть $X=\{x1,\ldots,xn\}$ — множество объектов, Y — множество ответов и имеется некоторая зависимость (детерминированная или вероятностная), позволяющая по $x \in X$ предсказать $y \in Y$,. $\{xn+1,\ldots,xm\}$ — множество неразмеченных объектов.

Результат обучения: *индикатор класса.*

Целью трансдуктивного обучения является выведение правильных меток только для неразмеченных объектов.

[5] https://intellect.icu/transduktivnoe-obuchenie-transduction-122

Трансдукцией называют выводы о частных случаях (тестовых данных) на основании частных случаев (данных обучения). Понятие трансдукции было введено Владимиром Вапником, трансдукция может быть отнесена к индукции, поскольку индукция требует решения общей задачи (восстановления функции) перед решением задачи более конкретной (вычисление результатов для новых объектов)[6].

Вапник: «При решении интересующей проблемы не решайте более общую проблему как промежуточный шаг. Постарайтесь получить ответ, который вам действительно нужен, но не более общий».

Такое обучение применимо на практике и в ситуациях, когда это множество не фиксировано, но изменяется достаточно медленно для того, чтобы при каждом изменении можно было подобрать модель заново. Трансдуктивное обучение является хорошим способом построения более качественных моделей в ситуациях, когда размер обучающей выборки явно недостаточен.

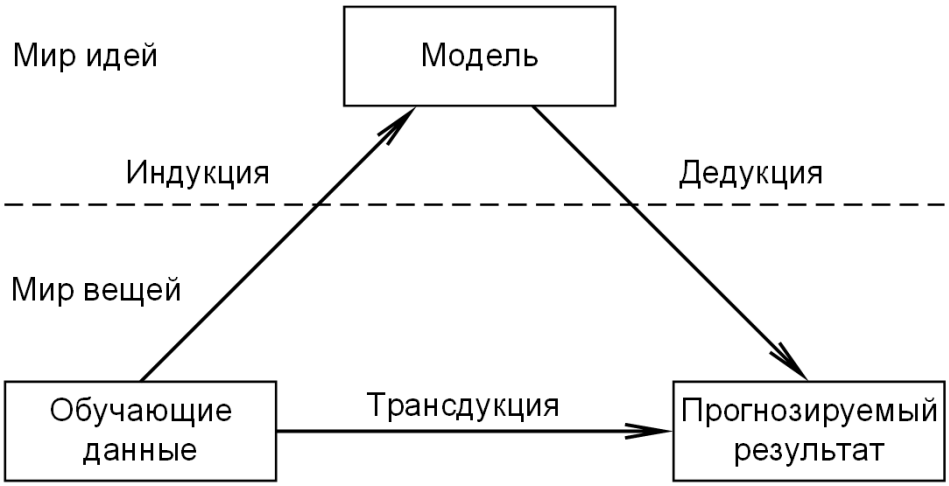


Рисунок 6 – Схема соотношения видов обучения[7]

Методы: трансдуктивная машина опорных векторов (Transductive Support Vector Machine, TSVM), Машина Байесовых Комитетов (Bayesian Committee Machine, BCM).

Обучение с подкреплением

- **Обучение с подкреплением (reinforcement learning) / Стимулируемое обучение**

Пусть имеется множество состояний окружения S, множество действий A и множество скалярных "выигрышей" R. В произвольный момент времени t агент характеризуется состоянием $s_t \in S$ и множеством возможных действий $A(s_t)$. Выбирая действие $a \in A(s_t)$, он переходит в состояние s_{t+1} и получает выигрыш r_t. Основываясь на таком взаимодействии с окружающей средой, агент, обучающийся с подкреплением,

[6] http://www.machinelearning.ru/wiki/index.php?title=%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%B4%D1%83%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5

[7] https://foobar167.github.io/page/vvedeniye-v-mashinnoye-obucheniye-i-iskusstvennyye-neyronnyye-seti.html

должен выработать стратегию $\pi: S \rightarrow A$, которая приведёт к максимальному выигрышу $R=r_0+r_1+\cdots+r_n$.

Результат: *тактика / стратегия.*

В этом случае правильных ответов не существует, а алгоритм пытается найти оптимальную стратегию.

Алгоритм обучается самостоятельно (на сырых данных), взаимодействуя с незнакомой средой и получая отклик на свои действия. Основная задача алгоритма — методом проб и ошибок выбрать те тактики, которые позволят максимизировать общую выгоду агента.

Этот тип машинного обучения требует использования системы вознаграждения/штрафа. Цель — вознаградить машину, когда она учится правильно, и наказать машину, когда она учится неправильно.

В обучении с подкреплением существует агент (*agent*) взаимодействует с окружающей средой (*environment*), предпринимая действия (*actions*). Окружающая среда даёт награду (*reward*) за эти действия, а агент продолжает их предпринимать.

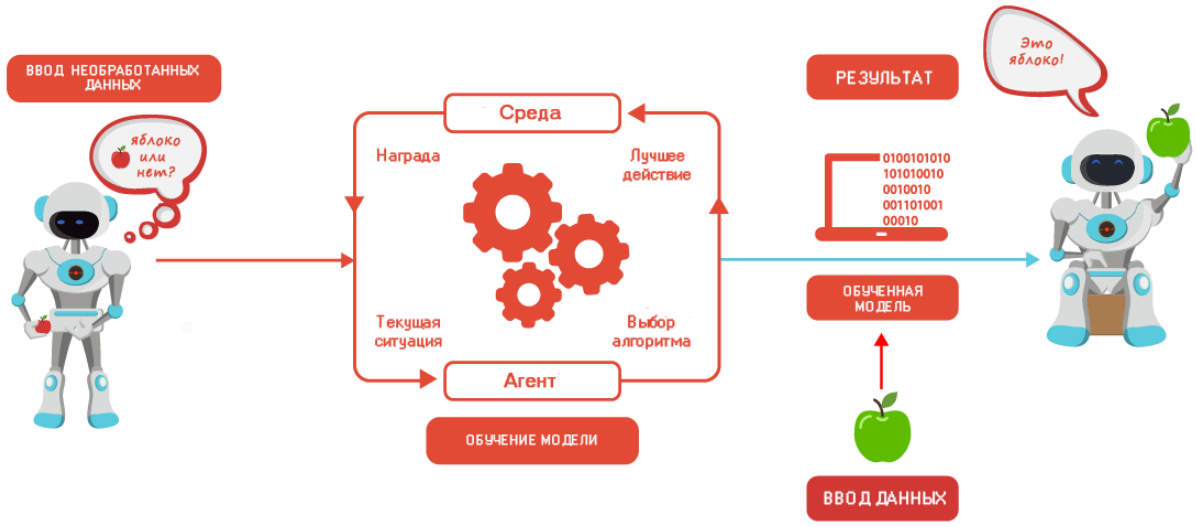


Рисунок 7 – Схема работы с подкреплением (со средой) [8]

Популярная тестовая среда для обучения с подкреплением — компьютерные игры, работающие по такому же принципу, баллы в игре - выигрыш.

Обучение с подкреплением особенно хорошо подходит для решения задач, связанных с выбором между долгосрочной и краткосрочной выгодой.

Динамическое обучение

- **Динамическое обучение (online learning)**

Динамическое обучение может быть как обучением с учителем, так и без учителя. Специфика в том, что прецеденты поступают потоком. Требуется немедленно принимать решение по каждому прецеденту и одновременно доучивать модель зависимости с учётом

[8] https://rb.ru/story/ai-dictionary/

новых прецедентов. Как и в задачах прогнозирования, здесь существенную роль играет фактор времени.

Активное обучение

- **Активное обучение (active learning)**

Похоже на обучение с учителем с той разницей, что ответы изначально неизвестны. Основная идея состоит в том, что алгоритм сам может обучаться на малых выборках, если он сам выбирает какие данные ему нужны[9]. То есть алгоритм составляет запросы, ответы на которые помогают ему обучаться. Основные стратегии: отбор объектов из выборки и из потока, синтез объектов.

Метаобучение

- **Метаобучение (meta-learning или learning-to-learn)**

Выделяют несколько видов метаобучения[10]:

– *Многозадачное обучение (multi-task learning)* одновременное обучение группы взаимосвязанных или схожих задач, для каждой из которых задаются свои пары «ситуация, требуемое решение», с помощью различных алгоритмов обучения, имеющих схожее внутренне представление. Информация о сходстве задач между собой позволяет более эффективно совершенствовать алгоритм обучения и повышать качество решения основной задачи.

– *Многовариантное обучение (multiple-instance learning)* обучение, когда прецеденты могут быть объединены в группы, в каждой из которых для всех прецедентов имеется «ситуация», но только для одного из них (причем, неизвестно какого) имеется пара «ситуация, требуемое решение», обычно использует набор данных, сгруппированных в виде пакетов. Метки пакетов известны, но неизвестны метки отдельных объектов. Задача — обучить модель для прогнозирования меток пакетов или меток объектов (или и тех и других). Обычно делается предположение, связывающее метки пакетов с отдельными экземплярами: если в пакете есть хотя бы один положительный объект, то пакет помечается как положительный. Если же все экземпляры отрицательные, то пакет считается отрицательным[11];

– *Индуктивный перенос (inductive transfer)* - опыт решения отдельных *частных* задач обучения по прецедентам переносится на решение последующих *частных* задач обучения.

Глубокое обучение

- **Глубокое обучение (deep learning)**

  Отличается от просто машинного обучения

  – количеством обрабатываемых данных,

[9] http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5

[10] http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5

[11] https://www.reg.ru/blog/deep-learning-in-aviation-safety/

– аппаратной зависимостью методов обучения,
– характеристикой получаемых результатов.

Машинное обучение работает с различными объемами данных и в основном используется для небольших объемов данных. Глубокое обучение, с другой стороны, работает эффективно, если объем данных быстро увеличивается.

Алгоритмы глубокого обучения предназначены для сильной зависимости от высокопроизводительных машин, в отличие от традиционных алгоритмов машинного обучения. Алгоритмы глубокого обучения выполняют большое количество операций умножения матриц, что требует огромной аппаратной поддержки.

Например, традиционные шаблоны машинного обучения фокусируются на пикселях и других атрибутах, необходимых для процесса разработки функций. Алгоритмы глубокого обучения фокусируются на высокоуровневых особенностях данных. Это уменьшает задачу разработки экстрактора новых функций для каждой новой проблемы.

Тема 2. Основные задачи машинного обучения.

В зависимости от вида обучения средствами ML могут решаться большой круг задач, примеров уже используемых решений на практике множество (Рисунок 8).

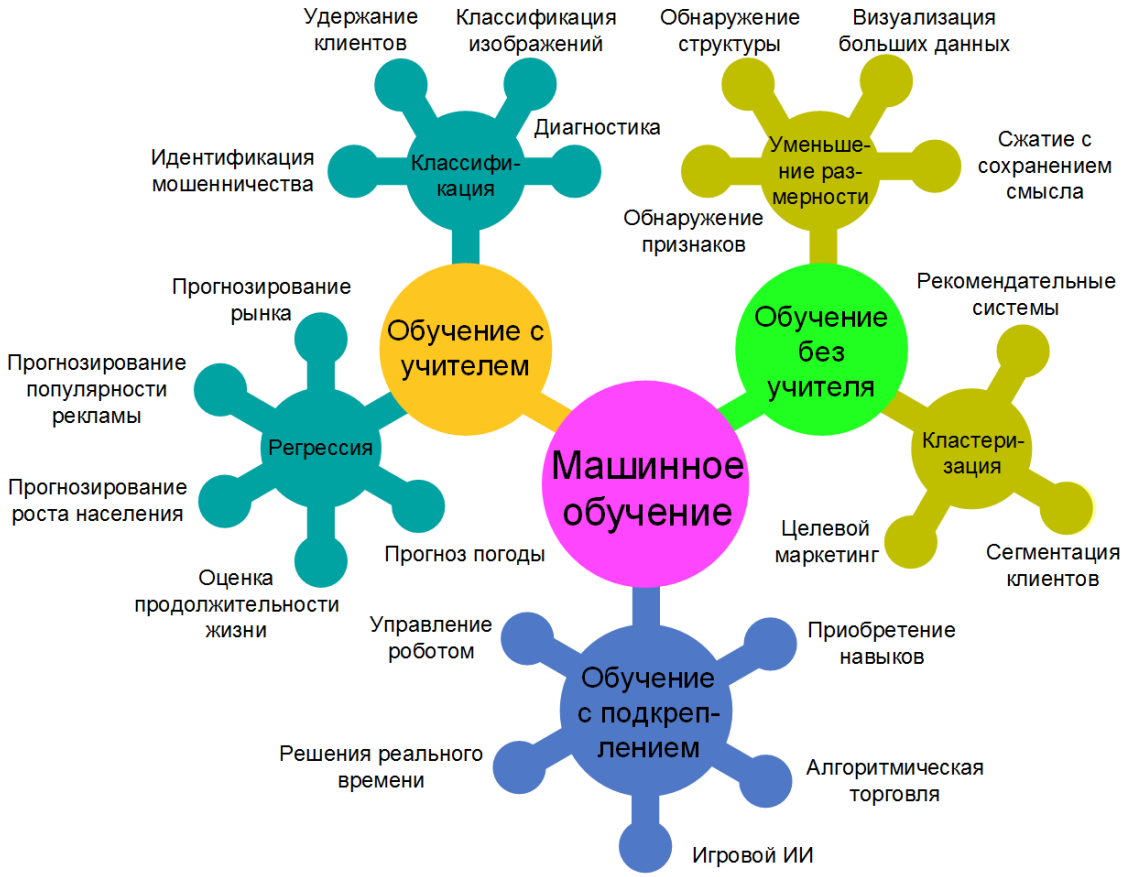


Рисунок 8 - Некоторые типовые задачи машинного обучения[12]

[12] https://foobar167.github.io/page/vvedeniye-v-mashinnoye-obucheniye-i-iskusstvennyye-neyronnyye-seti.html

Задачи обучения с учителем

Задача	**Описание**
классификация (classification)	множество допустимых ответов конечно. Их называют метками классов (class label). Класс — это множество всех объектов с данным значением метки.
регрессия (regression)	допустимым ответом является действительное число или числовой вектор.
ранжирование (learning to rank)	ответы надо получить сразу на множестве объектов, после чего отсортировать их по значениям ответов. Может сводиться к задачам классификации или регрессии. Часто применяется в информационном поиске и анализе текстов.
прогнозирования (forecasting)	объектами являются отрезки временных рядов, обрывающиеся в тот момент, когда требуется сделать прогноз на будущее. Для решения задач прогнозирования часто удаётся приспособить методы регрессии или классификации, причём во втором случае речь идёт скорее о задачах принятия решений.
выявление аномалий	на основании признаков научиться различать отличать аномалии от «не-аномалий». Кажется, что от задачи классификации эта задача ничем не отличается. Но особенность выявления аномалий состоит в том, что примеров аномалий для тренировки модели у нас либо очень мало, либо нет совсем, поэтому мы не можем решать такую задачу как задачу классификации.
структурное обучение (structured learning)	

Задачи обучения без учителя

Задача	**Описание**
	Задачи обучения без учителя
кластеризация (clustering)	сгруппировать объекты в кластеры, используя данные о попарном сходстве объектов. Функционалы качества могут определяться по-разному, например, как отношение средних межкластерных и внутрикластерных расстояний.
ассоциация или поиск	Исходные данные представляются в виде признаковых

ассоциативных правил (association rules learning).	описаний. Требуется найти такие наборы признаков, и такие значения этих признаков, которые особенно часто (неслучайно часто) встречаются в признаковых описаниях объектов
фильтрация выбросов (outliers detection)	обнаружение в обучающей выборке небольшого числа нетипичных объектов. В некоторых приложениях их поиск является самоцелью (например, обнаружение мошенничества). В других приложениях эти объекты являются следствием ошибок в данных или неточности модели, то есть шумом, мешающим настраивать модель, и должны быть удалены из выборки, см. также робастные методы и одноклассовая классификация.
построение доверительной области (quantile estimation)	области минимального объёма с достаточно гладкой границей, содержащей заданную долю выборки.
сокращение размерности (dimensionality reduction)	по исходным признакам с помощью некоторых функций преобразования перейти к наименьшему числу новых признаков, не потеряв при этом никакой существенной информации об объектах выборки. В классе линейных преобразований наиболее известным примером является метод главных компонент.
заполнение пропущенных значений (missing values)	замена недостающих значений в матрице объекты–признаки их прогнозными значениями.
визуализация	

Тема 3. Особенности сбора и обработки данных.

Загрузка данных для аналитической обработки (библиотека Pandas)

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.

Методы и атрибуты объектов приведены ниже (Таблица 1, Таблица 2).

Таблица 1 – Атрибуты

at	Доступ к одному значению для пары этикетка строки/столбца.
attrs	Словарь глобальных атрибутов данного набора данных.
axes	Возвращает список,представляющий оси DataFrame.
columns	Метки столбцов DataFrame.
dtypes	Верните стипы в DataFrame.
empty	Индикатор того,пуст ли DataFrame.
flags	Получение свойств,связанных с данным объектом pandas.
iat	Доступ к единственному значению для пары строка/столбцовая по целому числу позиций.
iloc	Индексирование на основе чисел для выбора по позициям.
index	Индекс (метки строк)DataFrame.
loc	Доступ к группе строк и столбцов по меткам или булевым массивам.
ndim	Возвращает int,представляющую количество осей/размеров массива.
shape	Возвращает кортеж,представляющий размерность DataFrame.
size	Возвращает int,представляющий количество элементов в этом объекте.
style	Возвращает объект Styler.
values	Возвращает Numpy представление DataFrame.

Таблица 2– Методы

abs()	Возврат Series/DataFrame с абсолютным числовым значением каждого элемента.
add (другое [, ось, уровень, fill_value])	Получить Сложение фрейма данных и другого,поэлементно (бинарный оператор*add*).
add_prefix(prefix)	Префиксные метки со строкой*prefix*.
add_suffix(suffix)	Суффиксные метки со строкой*suffix*.
agg([func, axis])	Агрегируйте,используя одну или несколько операций над указанной осью.
aggregate([func, axis])	Агрегируйте,используя одну или несколько операций над указанной осью.
align (другое [, соединение, ось, уровень, копия,…])	Выравнивание двух объектов по их осям с помощью указанного метода объединения.
all ([ось, bool_only, skipna, level])	Возвращает,все ли элементы верны,потенциально по оси.
any ([ось, bool_only, skipna, level])	Возвращает,является ли какой-либо элемент истинным,потенциально по оси.
append(other[, ignore_index, …])	Добавьте строки*other*до конца вызывающей стороны,возвращая новый объект.
apply (func [, ось, raw, result_type, args])	Применить функцию вдоль оси DataFrame.
applymap(func[, na_action])	Примените функцию к элементам Датафрейма.
asfreq (freq [, метод, как, нормализовать,…])	Преобразование временной серии в указанную частоту.
asof(where[, subset])	Возвращает последнюю(ие)строку(и)без NaNs перед*where*.
assign(**kwargs)	Присваивайте новые столбцы фрейму DataFrame.
astype (dtype [, копия, ошибки])	Приведите объект pandas к указанному dtype `dtype` .

at time (время [, asof, axis])	Выберите значения в определенное время суток (например,9:30 утра).
backfill ([ось, на месте, предел, вниз])	Синоним DataFrame.fillna() с method='bfill'.
between time (время_пуска , время_кончания [,…])	Выберите значения в определенное время суток (например,9:00-9:30 AM).
bfill ([ось, inplace, limit, downcast])	Синоним DataFrame.fillna() с method='bfill'.
bool()	Возвращает bool одного элемента Series или DataFrame.
boxplot ([столбец, по, топор, размер шрифта, гниль,…])	Сделайте график из столбцов DataFrame.
clip ([нижний, верхний, ось, на месте])	Обрезка значений при входном пороге (входных порогах).
combine (другое, func [, fill_value, overwrite])	Выполните комбинацию по столбцам с другим DataFrame.
combine first(other)	Обновление нулевых элементов со значением в том же месте в*other*.
compare (другие [, align_axis, keep_shape,…])	Сравните с другим DataFrame и покажите различия.
convert dtypes([infer_objects, …])	Преобразовать столбцы в наилучших dtypes с использованием dtypes поддерживающих pd.NA.
copy([deep])	Сделайте копию индексов и данных этого объекта.
corr([method, min_periods])	Вычислите парную корреляцию столбцов,исключая значения NA/ноль.
corrwith(other[, axis, drop, method])	Вычислите парную корреляцию.
count ([ось, уровень, только число])	Считайте не-НК клетки для каждого столбца или строки.
cov([min_periods, ddof])	Вычислите парные ковариационные столбцы,исключая NA/нуклевые значения.
cummax([axis, skipna])	Возврат кумулятивного максимума по оси DataFrame или Series.
cummin([axis, skipna])	Возврат кумулятивного минимума по оси DataFrame или Series.
cumprod([axis, skipna])	Возврат кумулятивного продукта по оси DataFrame или Series.
cumsum([axis, skipna])	Возврат кумулятивной суммы по оси DataFrame или Series.
describe ([процентили, включить, исключить,…])	Создайте описательную статистику.
diff([periods, axis])	Первое дискретное различие элементов.
div (другой [, ось, уровень, fill_value])	Получить плавающее деление фрейма данных и других,поэлементно (двоичный оператор*truediv*).
divide (другой [, ось, уровень, fill_value])	Получить плавающее деление фрейма данных и других,поэлементно (двоичный оператор*truediv*).
dot(other)	Вычислите умножение матриц между DataFrame и другими.
drop ([метки, ось, индекс, столбцы, уровень,…])	Сбросьте указанные метки со строк или столбцов.
drop duplicates ([подмножество, сохранить,	Возвращает DataFrame с удаленными дублирующимися строками.

на месте,…])	
droplevel(level[, axis])	Возвращает DataFrame с удаленным уровнем(ами)запрашиваемого индекса/столбца.
dropna ([ось, как, порог, подмножество, на месте])	Удалить отсутствующие значения.
duplicated([subset, keep])	Возвращает булевы серии,обозначающие дублирующиеся строки.
eq (прочее [, ось, уровень])	Получение равенства между фреймом данных и другим,поэлементно (бинарный оператор*eq*).
equals(other)	Проверьте,содержат ли два объекта одни и те же элементы.
eval(expr[, inplace])	Оцените строку,описывающую операции со столбцами DataFrame.
ewm([com, span, halflife, alpha, …])	Предоставьте экспоненциальные взвешенные (EW)функции.
expanding([min_periods, center, axis])	Обеспечить расширяющиеся преобразования.
explode(column[, ignore_index])	Преобразуйте каждый элемент списка в строку,дублируя значения индексов.
ffill ([ось, inplace, limit, downcast])	Синоним DataFrame.fillna() с method='ffill'.
fillna ([значение, метод, ось, inplace,…])	Заполните значения NA/NaN указанным методом.
filter ([элементы, вроде, регулярное выражение, ось])	Подмножество строк или столбцов датафрейма в соответствии с указанными индексными метками.
first(offset)	Выбор начальных периодов данных временного ряда на основе смещения даты.
first_valid_index()	Индекс возврата для первого значения,отличного от нулевого.
floordiv (другой [, ось, уровень, fill_value])	Получить целочисленное деление фрейма данных и другого,поэлементно (двоичный оператор*floordiv*).
from_dict (данные [, ориентация, тип данных, столбцы])	Постройте DataFrame из диктата массивов или диктатов.
from_records (данные [, индекс, исключить,…])	Преобразование структурированных или записывающих файлов в DataFrame.
ge (другой [, ось, уровень])	Получить значение "Больше или равно" из фрейма данных и других,поэлементно (бинарный оператор*ge*).
get(key[, default])	Получить элемент из объекта для данного ключа (например:столбец DataFrame).
groupby ([по, оси, уровню, as_index, sort,…])	Группировка DataFrame с помощью сопоставителя или по серии столбцов.
gt (другой [, ось, уровень])	Получение большего числа фреймов данных и других,по элементам (бинарный оператор*gt*).
head([n])	Верните первый*n*rows.
hist ([столбец, по, сетка, xlabelsize, xrot,…])	Сделайте гистограмму DataFrame.
idxmax([axis, skipna])	Индекс возврата первого максимума по запрашиваемой оси.
idxmin([axis, skipna])	Индекс возврата первого появления минимума по запрошенной оси.
infer_objects()	Попробуйте прийти к выводу,что для столбцов объектов

	лучше подходят стили.
info ([подробный, buf, max_cols, memory_usage,…])	Распечатайте краткое резюме DataFrame.
insert(loc, column, value[, allow_duplicates])	Вставьте колонку в DataFrame в указанном месте.
interpolate ([метод, ось, предел, на месте,…])	Заполнение значений NaN методом интерполяции.
isin(values)	Содержится ли каждый элемент в DataFrame в значениях.
isna()	Обнаружить недостающие значения.
isnull()	Обнаружить недостающие значения.
items()	Итерация по парам (название колонки,серия).
iteritems()	Итерация по парам (название колонки,серия).
iterrows()	Итерация по строкам DataFrame в виде пар (индекс,серия).
itertuples([index, name])	Итерация по строкам DataFrame в виде именных фреймов.
join (другие [, на, как, lsuffix, rsuffix, sort])	Присоедините колонки другого DataFrame.
keys()	Получите «информационную ось» (подробнее см. Индексирование).
kurt ([ось, скипна, уровень, только число])	Возвращает несмещенный эксцесс по заданной оси.
kurtosis ([ось, скипна, уровень, только число])	Возвращает несмещенный эксцесс по заданной оси.
last(offset)	Выбор конечных периодов данных временного ряда на основе смещения даты.
last valid index()	Индекс возврата для последнего значения,отличного от нулевого.
le (другой [, ось, уровень])	Получить Less than or equal to из dataframe и other,по элементам (бинарный оператор*le*).
lookup(row_labels, col_labels)	(УСТАРЕЛО) Функция «причудливой индексации» на основе меток для DataFrame.
lt (другой [, ось, уровень])	Получить значение Less than из фрейма данных и другого,поэлементно (бинарный оператор*lt*).
mad ([ось, скипна, уровень])	{desc}
mask(cond[, other, inplace, axis, level, …])	Замените значения там,где условие истинно.
max ([ось, skipna, level, numeric_only])	Возвращает максимальное из значений по заданной оси.
mean ([ось, скипна, уровень, только число])	Возвращает среднее значение значений по заданной оси.
median ([ось, skipna, level, numeric_only])	Возвращает медиану значений по заданной оси.
melt([id_vars, value_vars, var_name, …])	Разворачивает фрейм данных из широкого формата в длинный,по желанию оставляя установленные идентификаторы.
memory usage([index, deep])	Возвращает использование памяти каждого столбца в байтах.
merge (right [, как, on, left_on, right_on,…])	Объедините DataFrame или именованные объекты Series с соединением в стиле базы данных.
min ([ось, skipna, level, numeric_only])	Возвращает минимальное из значений по заданной оси.

mod (другой [, ось, уровень, fill_value])	Получить модуляцию фрейма данных и других,поэлементно (бинарный оператор*mod*).
mode ([ось, numeric_only, dropna])	Получить режим(ы)каждого элемента вдоль выделенной оси.
mul (другой [, ось, уровень, fill_value])	Получить умножение фрейма данных и другого,поэлементно (бинарный оператор*mul*).
multiply (другой [, ось, уровень, fill_value])	Получить умножение фрейма данных и другого,поэлементно (бинарный оператор*mul*).
ne (другой [, ось, уровень])	Получить Не равно из фрейма данных и другого,поэлементно (бинарный оператор*ne*).
nlargest (n, столбцы [, сохранить])	Верните первый*n*строки,упорядоченные по*columns*в порядке убывания.
notna()	Обнаружение существующих (не пропущенных)значений.
notnull()	Обнаружение существующих (не пропущенных)значений.
nsmallest(n, columns[, keep])	Верните первый*n*строки,упорядоченные по*columns*в порядке возрастания.
nunique([axis, dropna])	Посчитайте отдельные наблюдения по запрашиваемой оси.
pad ([ось, inplace, limit, downcast])	Синоним DataFrame.fillna() с method='ffill'.
pct_change ([периоды, fill_method, limit, freq])	Изменение в процентах между текущим и предыдущим элементом.
pipe(func, *args, **kwargs)	Применить функционал(self,*args,**kwargs).
pivot ([индекс, столбцы, значения])	Возврат переформатированного DataFrame,организованного по заданным значениям индекса/столбца.
pivot_table ([значения, индекс, столбцы,…])	Создайте поворотную таблицу в стиле электронных таблиц в виде DataFrame.
plot	псевдоним pandas.plotting._core.PlotAccessor
pop(item)	Возвратить изделие и выпасть из рамы.
pow (другой [, ось, уровень, fill_value])	Получить экспоненциальную мощность фрейма данных и других,поэлементно (бинарный оператор*pow*).
prod ([ось, skipna, level, numeric_only,…])	Возвращает произведение значений по заданной оси.
product ([ось, skipna, level, numeric_only,…])	Возвращает произведение значений по заданной оси.
quantile ([q, ось, только число, интерполяция])	Возврат значений в заданном квантиле по запрашиваемой оси.
query(expr[, inplace])	Запрос столбцов DataFrame с булевым выражением.
radd (другое [, ось, уровень, значение_заполнения])	Получить Сложение фрейма данных и другого,поэлементно (бинарный оператор*radd*).
rank ([ось, метод, numeric_only,…])	Вычислите числовые ряды данных (от 1 до n)вдоль оси.
rdiv (другой [, ось, уровень, fill_value])	Получить плавающее деление фрейма данных и других,поэлементно (двоичный оператор*rtruediv*).
reindex ([метки, индекс, столбцы, ось,…])	Приведите серию/кадр данных в соответствие с новым индексом с помощью дополнительной логики заполнения.
reindex_like (другой [, метод, копия, ограничение,…])	Возвращает объект с соответствующими индексами как другой объект.
rename ([сопоставитель,	Ярлыки с переменными осями.

индекс, столбцы, ось, копия,…])	
rename_axis([преобразователь, индекс, столбцы, ось,…])	Задайте название оси для индекса или столбцов.
reorder_levels(order[, axis])	Уровни индексов заднего ряда с использованием порядка ввода.
replace([to_replace, значение, inplace, limit,…])	Замените значения,указанные в*to_replace*with*value*.
resample(правило [, ось, закрыто, метка,…])	Перепробовать данные временных рядов.
reset_index([уровень, падение, место,…])	Сбросьте индекс,или его уровень.
rfloordiv(другой [, ось, уровень, fill_value])	Получить целочисленное деление фрейма данных и другого,поэлементно (двоичный оператор*rfloordiv*).
rmod(другой [, ось, уровень, fill_value])	Получить модуляцию фрейма данных и других,поэлементно (бинарный оператор*rmod*).
rmul(другой [, ось, уровень, fill_value])	Получить умножение фрейма данных и другого,поэлементно (бинарный оператор*rmul*).
rolling(окно [, min_periods, центр, ...])	Предоставить расчеты по скользящим окнам.
round([decimals])	Округлите DataFrame до переменного числа десятичных разрядов.
rpow(другой [, ось, уровень, fill_value])	Получить экспоненциальную мощность фрейма данных и других,поэлементно (бинарный оператор*rpow*).
rsub(другой [, ось, уровень, fill_value])	Получить вычитание фрейма данных и другого,поэлементно (бинарный оператор*rsub*).
rtruediv(другой [, ось, уровень, fill_value])	Получить плавающее деление фрейма данных и других,поэлементно (двоичный оператор*rtruediv*).
sample([n, frac, replace, weights, …])	Возврат случайной выборки элементов с оси объекта.
select_dtypes([include, exclude])	Возвращает подмножество столбцов DataFrame на основе dtypes столбцов.
sem([ось, skipna, level, ddof, numeric_only])	Возврат несмещенной стандартной ошибки среднего по запрашиваемой оси.
set_axis(метки [, ось, на месте])	Назначьте нужный индекс на данную ось.
set_flags(* [, копировать, allow_duplicate_labels])	Возвращает новый объект с обновленными флагами.
set_index(keys[, drop, append, inplace, …])	Установите индекс DataFrame,используя существующие столбцы.
shift([периоды, частота, ось, значение_заполнения])	Сдвиг индекса на желаемое количество периодов с дополнительным временем*freq*.
skew([ось, скипна, уровень, только число])	Возвращает несмещенный перекос по заданной оси.
slice_shift([periods, axis])	(УСТАРЕЛО) Эквивалентно*shift*без копирования данных.
sort_index([ось, уровень, возрастание,…])	Сортируйте объект по меткам (по оси).
sort_values(по [, оси, по возрастанию, по месту,…])	Сортируйте по значениям по обеим осям.
sparse	псевдоним

	pandas.core.arrays.sparse.accessor.SparseFrameAccessor
squeeze([axis])	Сожмите объекты 1 размерной оси в скаляры.
stack([level, dropna])	Сложите предписанный уровень(ы)из столбцов в индекс.
std ([ось, skipna, level, ddof, numeric_only])	Вернуть образец со стандартным отклонением по запрашиваемой оси.
sub (другой [, ось, уровень, fill_value])	Получить вычитание фрейма данных и другого,поэлементно (бинарный оператор*sub*).
subtract (другое [, ось, уровень, fill_value])	Получить вычитание фрейма данных и другого,поэлементно (бинарный оператор*sub*).
sum ([ось, skipna, level, numeric_only,…])	Возвращает сумму значений по заданной оси.
swapaxes (ось1, ось2 [, копия])	Оси взаимозаменяются,а оси значений взаимозаменяются соответствующим образом.
swaplevel([i, j, axis])	Замена уровней i и j в MultiIndex на определенную ось.
tail([n])	Верните последний*n*rows.
take (индексы [, ось, is_copy])	Возвращает элементы с указанными *позиционными* индексами вдоль оси.
to_clipboard([excel, sep])	Скопируйте объект в системный буфер обмена.
to_csv ([путь_или_буф, sep, na_rep,…])	Запишите объект в файл с разделенными запятыми значениями (csv).
to_dict([orient, into])	Преобразование DataFrame в словарь.
to_excel(excel_writer[, sheet_name, na_rep, …])	Запишите объект на лист Excel.
to_feather(path, **kwargs)	Запись DataFrame в двоичный формат Feather.
to_gbq (таблица_назначения [, идентификатор_проекта,…])	Запишите DataFrame в таблицу Google BigQuery.
to_hdf (путь_или_буф, ключ [, режим, завершение,…])	Запишите содержащиеся данные в файл HDF5 с помощью HDFStore.
to_html([buf, columns, col_space, header, …])	Рендеринг DataFrame как HTML-таблицы.
to_json([path_or_buf, orient, date_format, …])	Преобразование объекта в JSON строку.
to_latex([buf, columns, col_space, header, …])	Вернуть объект в табличную,длинную или вложенную таблицу/таблицу LaTeX.
to_markdown([buf, mode, index, storage_options])	Печать DataFrame в формате,удобном для Markdown.
to_numpy([dtype, copy, na_value])	Преобразование DataFrame в массив NumPy.
to_parquet ([путь, двигатель, сжатие,…])	Запишите DataFrame в двоичный формат паркета.
to_period ([частота, ось, копия])	Преобразование DataFrame из DatetimeIndex в PeriodIndex.
to_pickle (путь [, сжатие, протокол,…])	Объедините (сериализуйте)объект в файл.
to_records ([индекс, тип_колонки ,_дтип_индекса])	Преобразование DataFrame в массив записей NumPy.
to_sql (имя, con [, схема,	Запись записей,хранящихся в DataFrame,в базу данных

if_exists,…])	SQL.
to_stata (путь [, convert_dates, write_index,…])	Экспорт объекта DataFrame в формат Stata dta.
to_string([buf, columns, col_space, header, …])	Отправьте DataFrame в удобный для консоли табличный вывод.
to_timestamp ([частота, как, ось, копировать])	Приведено к DatetimeIndex меток времени в *начале* периода.
to_xarray()	Возвратите рентгеновский объект от панд.
transform(func[, axis])	Вызов `func` для самостоятельного создания DataFrame с преобразованными значениями.
transpose(*args[, copy])	Транспонируйте индекс и столбцы.
truediv (прочее [, ось, уровень, fill_value])	Получить плавающее деление фрейма данных и других,поэлементно (двоичный оператор*truediv*).
truncate ([до, после, ось, копировать])	Усечение серии или DataFrame до и после некоторого значения индекса.
tshift ([периоды, частота, ось])	(УСТАРЕЛО) Сдвиньте временной индекс, используя частоту индекса, если таковая имеется.
tz_convert (tz [, ось, уровень, копия])	Преобразование оси tz-aware в целевой часовой пояс.
tz_localize (tz [, ось, уровень, копия,…])	Локализация tz-наивного индекса серии или DataFrame в целевой временной зоне.
unstack([level, fill_value])	Поворот одного уровня (обязательно иерархического)индексных меток.
update (другое [, присоединиться, перезаписать,…])	Модифицируйте на месте,используя не-NA значения из другого DataFrame.
value_counts ([подмножество, нормализация, сортировка,…])	Возвращает серию,содержащую подсчет уникальных строк в DataFrame.
var ([ось, skipna, level, ddof, numeric_only])	Возвращение непредвзятой дисперсии по запрашиваемой оси.
where(cond[, other, inplace, axis, level, …])	Замените значения там,где условие False.
xs (ключ [, ось, уровень, drop_level])	Возврат поперечного сечения из Series/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')
```
Полный список параметров можно посмотреть https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html.
Pandas поддерживает загрузку данных и из других форматов (Таблица 3).

Таблица 3- Поддерживаемые форматы для загрузки данных и методыPandas

Тип данных	Формат данных	Используемый метод
Текстовый	CSV	read_csv
Текстовый	Fixed-Width Text File	read_fwf
Текстовый	JSON	read_json
Текстовый	HTML	read_html
Текстовый	Буфер обмена	read_clipboard
Бинарный	MS Excel	read_excel
Бинарный	OpenDocument	read_excel
Бинарный	HDF5 Format	read_hdf
Бинарный	Feather Format	read_feather
Бинарный	Parquet Format	read_parquet
Бинарный	ORC Format	read_orc

Бинарный	Msgpack	read_msgpack
Бинарный	Stata	read_stata
Бинарный	SAS	read_sas
Бинарный	SPSS	read_spss
Бинарный	Python Pickle Format	read_pickle
SQL	SQL	read_sql
SQL	Google BigQuery	read_gbq

Загрузка данных в https://colab.research.google.com/

Для загрузки данных из файла в среду выполнения Google CoLab можно воспользоваться библиотекой io.

Пример:

```
import pandas as pd
import io
from google.colab import files
uploaded = files.upload()
# выбрать после этого файл на диске и подождать загрузки
df2 = pd.read_csv(io.BytesIO(uploaded['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.

Более подробно о параметрах метода см. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html.

Пример:

```
rating[:10].to_csv('saved_ratings.csv', index=False)
```

Pandas поддерживает выгрузку данных в разные форматы (Таблица 4).

Таблица 4 - Поддерживаемые форматы для выгрузку данных и методыPandas

Тип данных	**Формат данных**	**Используемый метод**
Текстовый	CSV	to_csv
Текстовый	JSON	to_json
Текстовый	HTML	to_html
Текстовый	Буфер обмена	to_clipboard

Бинарный	MS Excel	to_excel
Бинарный	HDF5 Format	to_hdf
Бинарный	Feather Format	to_feather
Бинарный	Parquet Format	to_parquet
Бинарный	Msgpack	to_msgpack
Бинарный	Stata	to_stata
Бинарный	Python Pickle Format	to_pickle
SQL	SQL	to_sql
SQL	Google BigQuery	to_gbq

Вывод информации о наборе данных

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 (Таблица 5) значений фрейма используют метод describe():

```
fix_df.describe(include='all')
```

Таблица 5 – Вывод метода describe

count	Подсчёт частоты того или иного события (сколько раз произошло событие?).

mean	Среднее значение.
std	Стандартное отклонение (числовое значение, которое отображает изменение пределов данных).
min	Наименьшее число в наборе данных.
25%	25-й процентиль.
50%	50-й процентиль.
75%	75-й процентиль.
max	Максимальное число в наборе данных.

Для фильтрации данных необходимо сформировать условие (условие накладывается на столбцы) и указать его внутри квадратных скобок DataFrame[условие], например:

```
fix_df[fix_df['Berri 1'] > 6500]
```

Для получения количества уникальных значений в столбце можно использовать метод unique():

```
# Подсчет уникальных значений в столбце
len(fix_df['Berri 1'].unique())
```

Изменение набора данных

При работе с набором данных требуется:

- изменение данных:
  - изменение существующих значений в столбцах,
  - удаление и добавление строк,
  - обработка пустых значений и дубликатов,
- изменение столбцов:
  - добавление и удаление столбца,
  - ….

Изменение данных

Изменение значений столбца

Изменить значение можно с помощью явного присвоения (=) или с помощью метода apply(). Чтобы изменить данные с учётом некоторых условий, можно использовать вызов функции или лямбда-функции.

Пример:

```
df[‘romanic’].apply(lambda x: 1 if x == ‘yes’ else 0)
df[‘maximum’] = df[[‘c1’,’c2']].max(axis =1)
```

Преобразовать данные можно также с помощью метода map(), необходимо определить словарь, в котором «ключами» являются старые значения, а «значениями» – новые значения.

Пример:

```
level_map = {1: 'high', 2: 'medium', 3: 'low'}
df['c_level'] = df['c'].map(level_map)
```

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.

Подробнее см. https://pandas.pydata.org/docs/reference/api/pandas.isnull.html.

Пример:

```
#заполнение нулями
df.fillna(0)
#заполнение нулями (изменяем исходный)
df.fillna(0, inplace=True)
#заполнение средним значением
df.fillna(df.mean())
data.Age [data ['Age']. isnull ()] = age_mean
# Для Установите разные значения отсутствующей заливки для разных столбцов
df.fillna (value = {'id': 0, 'name': 'No', 'пол': 'unknown', 'age': 'unknown', 'weight': 0}, inplace = True)
# Заполнить себя значением предыдущей строки
df.fillna (method = 'ffill', inplace = True)
# Заполняем себя значением следующей строки
df.fillna (method = 'bfill', inplace = True)
# Используйте экстремальное значение, среднее, медианное и т. д. из разных столбцов для пропущенных
df.fillna ({'age': df.age.mean (), 'weight': df.weight.max ()}, inplace = True)
```

Подробнее см. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html.

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)
```

Подробнее см. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html.

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()
```

Визуализация данных

Визуализация данных в машинном обучении является отдельной важной задачей. Благодаря наглядному представлению, можно сделать определить вид распределения, наличие выбросов, виды зависимостей между признаками и др. Визуализация данных без ее интерпретации может нести мало информации, поэтому необходимо понимать, какую информацию можно извлечь или продемонстрировать в каждом виде визуализации.

Графики зависимости от временной переменной дают представление о динамике изменений значений признака и виде зависимости данных от временного периода.

Графики зависимости одной переменной от другой или диаграммы рассеивания демонстрируют наличие или отсутствие связь между признаками и вид этой связи.

Гистограммой (Рисунок 9) называют двухмерный график, по горизонтальной оси которого откладываются переменные или числовые интервалы, а по вертикальной – частота появления переменной (в заданном интервале).

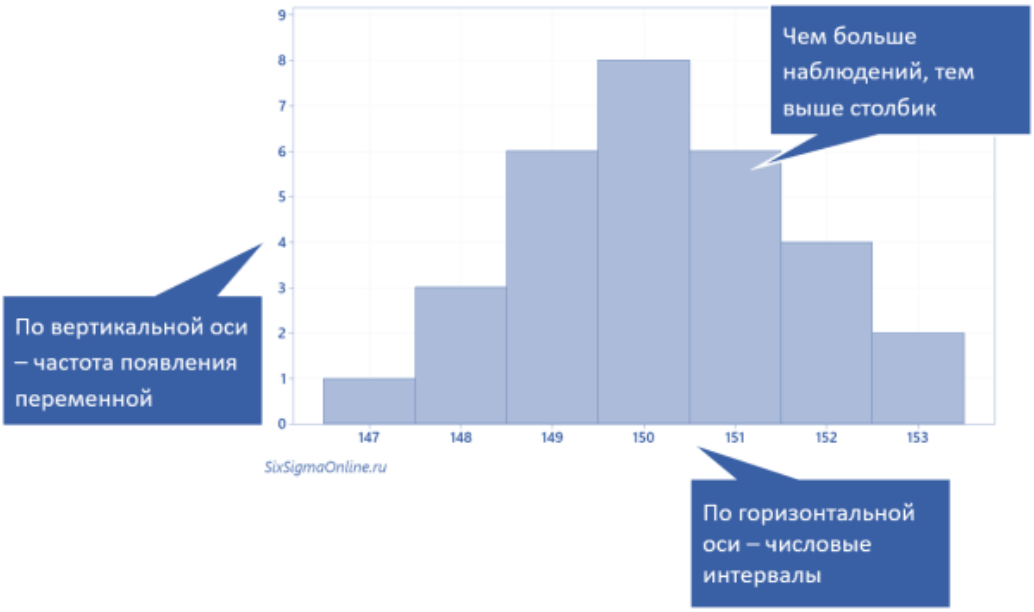


Рисунок 9 – Гистограмма

Визуально гистограмма похожа на столбчатую диаграмму, но они строятся разными способами (Рисунок 10).

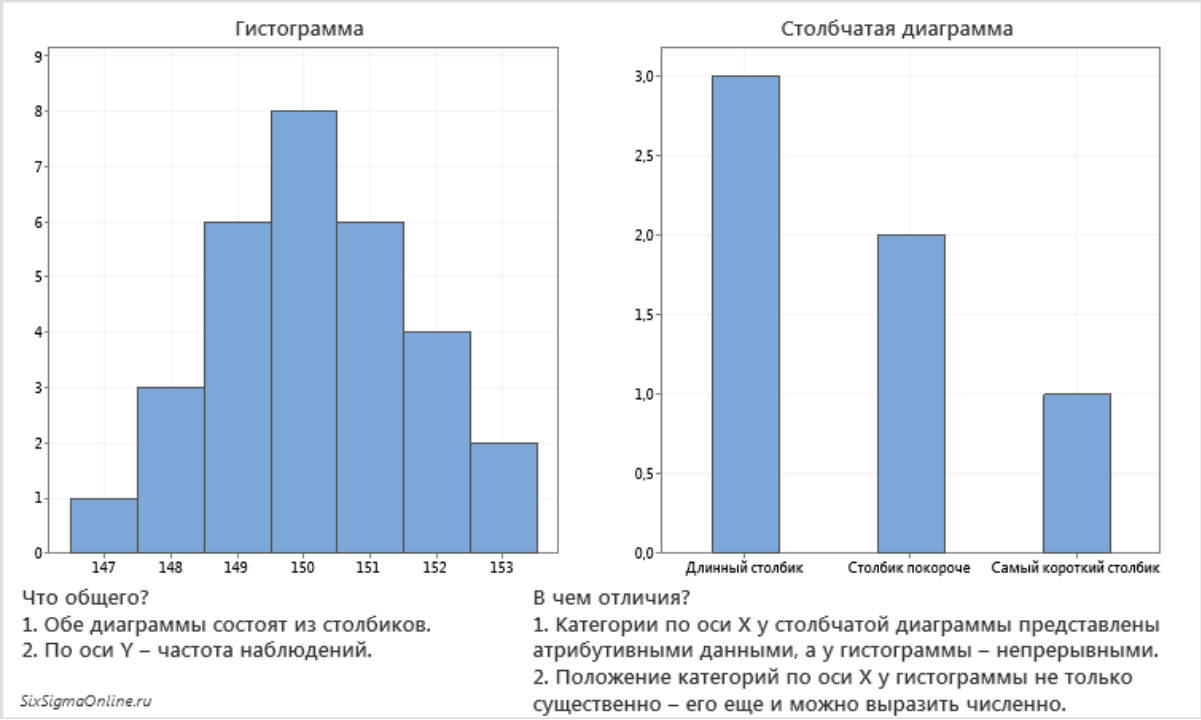


Рисунок 10 – Разница между столбчатой диаграммой и гистограммой

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.

Диаграмма размаха показывает данные о квантилях и выбросах (Рисунок 11).

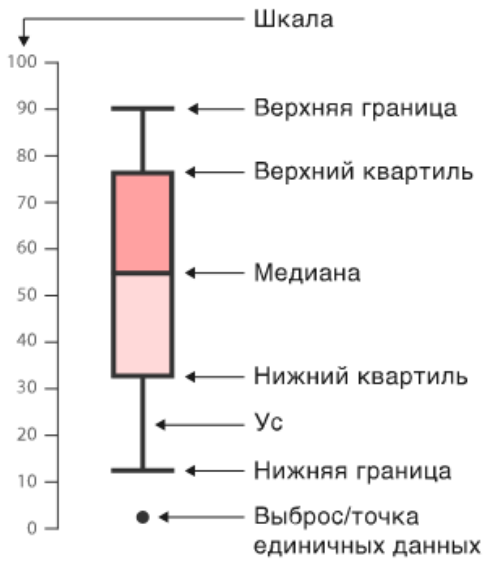


Рисунок 11 – Схеме диаграммы размаха

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()
```

Ниже представлены примеры основных видов графиков и диаграмм (Таблица 6).

Таблица 6 – Варианты визуализации данных

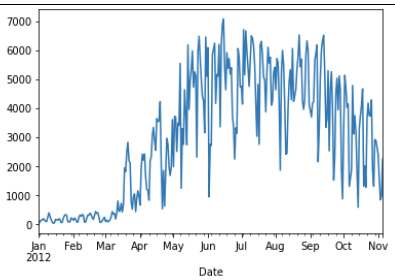
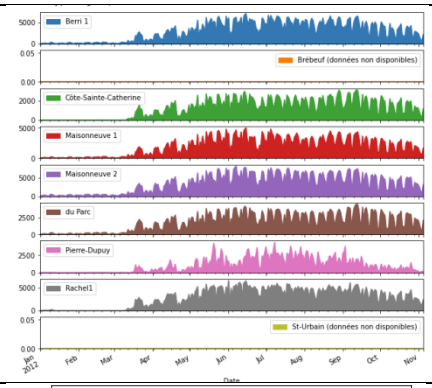
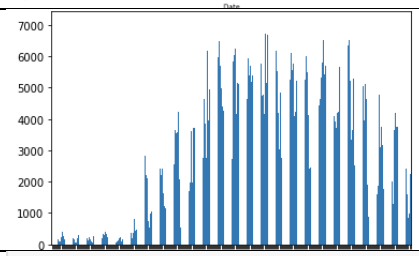
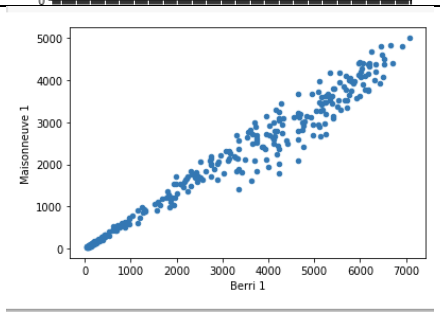
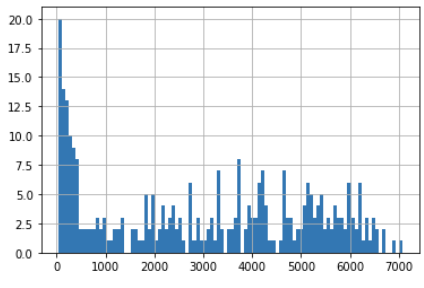
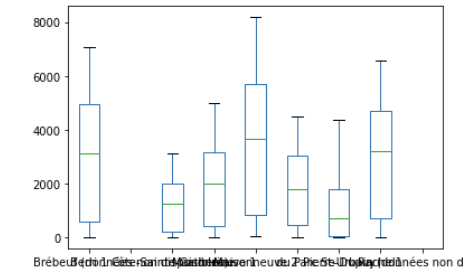
Тип графика	**Описание**	**Вид**	**Пример**
Линейный график	Линейный график строит переход от точки к точке.		`df2['Berri 1'].plot()`

Диаграмма площадей (областей)			`df2.plot.area(figsize=(22, 22), subplots=True)`
Столбиковая диаграмма	На столбиковой диаграмме каждая категория в виде столбика имеет высоту, соответствующую числовому значению этой категории.		`df2['Berri 1'].plot.bar()`

Диаграммы рассеивания	В диаграмме рассеяния каждая точка одного атрибута соответствует каждой точке другого.		`df2.plot.scatter(x='Berri 1', y='Maisonneuve 1');`
			
			`df2.plot(x='Berri 1', y='Maisonneuve 1', kind='scatter')`

Гистограмма	Количественные соотношения некоторого показателя представлены в виде прямоугольников, площади которых пропорциональны.		`df2['Berri 1'].hist();`
			
			`df2['Berri 1'].hist(bins=100);`

Диаграммы размаха («ящик с усами»)	способ визуального представления групп числовых данных через квартили.		`df2.plot.box();`
			
			`df2.boxplot(column='Berri 1')`

Пример:

```
df = pd.read_csv('some.csv')
# если Вам заранее известны типы, то чрезвычайно полезно задать типы колонок сразу - это сэкономит оперативную память.
# также можно использоть лишь несколько колонок, а не все колонки в файле
df = pd.read_csv('some.csv', usecols=["user_id", "id3"], dtype={"user_id": np.int64, "id3": np.uint16})
```

```
# пример создания объекта дата-фрейм
pd.DataFrame(columns=['A', 'B'])
# пример добавления строки в дата-фрейм
df.append({'A':1, 'B':2}, ignore_index=True)
df.head()       # первые строки
df.tail()       # последние строки
df.sample(5)    # случайно выбранное кол-ва строк, полезно использовать для
уменьшения матрицы для прогонки тестов
df.shape        # по аналогии с numpy - размерность матрицы
df.describe()   # математические данные
df.info()       # использование памяти
some_value = df.ix[2, 'som_col']  # похожие функции loc и iloc
filtered_data = df[df.some_col == 'apple']
filtered_data = df[(df.price > 10.0) & (df.some_col == 'apple')]
# пример удаления колонки:
df = df.drop("A", axis=1)
# удалит дубликаты
df = df.drop_duplicates()
# Создание новой колонки со значением по умолчанию
df['new_col'] = 1
# а вот так можно добавлять колонку с условием
df['new_col2'] = np.where(df['score']>=10.0, True, False)
def get_score(s):
    return s * 0.21
df['score'] = df.some_col.apply(get_score)
# переименовываем колонку A на колонку C
df = df.rename(columns={'A': 'C'})
# сортировка по одной колонке
df = df.sort_values('price', axis=0, ascending=False)
# сортировка сразу по нескольким колонкам
df = df.sort_values(['price', 'score'], ascending=[1, 0])
df = pd.concat([df1, df2], ignore_index=True, axis=0)
# группировка в дата-фрейме без мультииндекса
df.groupby(['A'], as_index=False).agg({'B': lambda series: series.iloc[0]})
# выгрузить в другом формате или сохранить до следующего раза новый фрагмент
таблицы дата-фрейма:
# в numpy массив
df.values
# в numpy массив, но сразу всю матрицу
df.as_matrix
# сохранить в файл
df.to_csv('submission.csv', index=False)
```

Тема 4. Задача классификации. Метод kNN, деревья решений, логистическая регрессия, SVM.

Общее описание библиотеки Scikit-learn

Библиотека Scikit-learn — содержит различные методы машинного обучения в том числе и методы обучения с учителем. Библиотека базируется на таких библиотеках как (Рисунок 12):

– **NumPy**: математические операциии и операции над тензорами,

– **SciPy**: научно-технические вычисления,

– **Matplotlib**: визуализация данных,

– **IPython**: интерактивная консоль для Python,

– **SymPy**: символьная математика,

– **Pandas**: обработка, манипуляции и анализ данных.

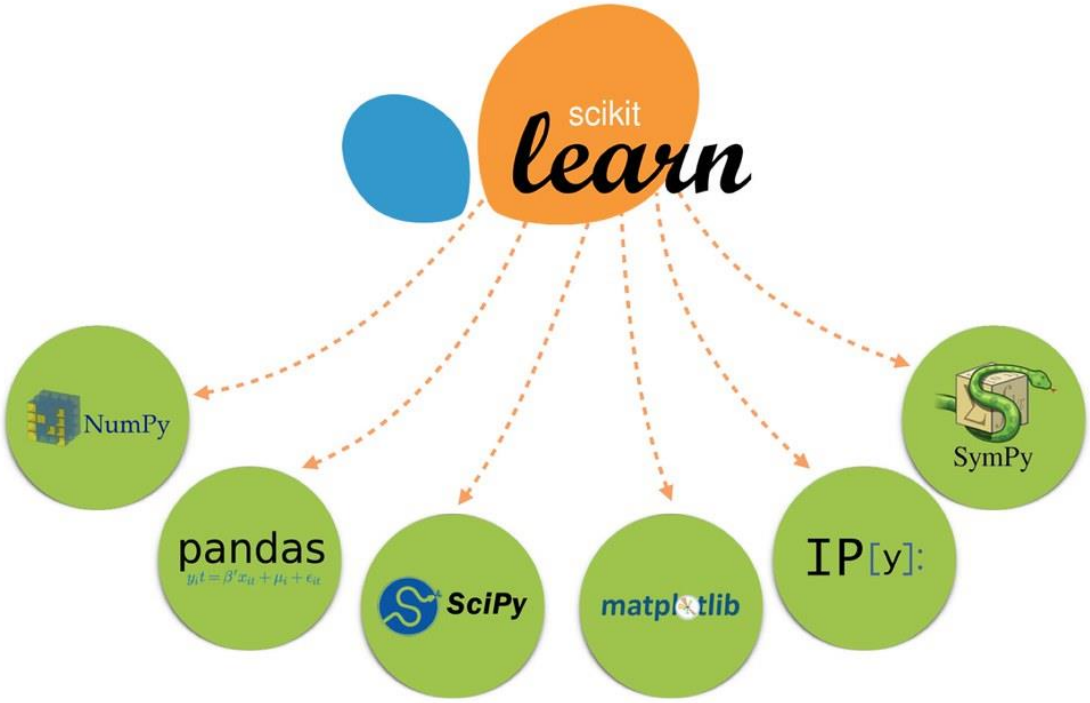


Рисунок 12 – Базовые библиотеки для Scikit-learn

Scikit-learn поддерживается широким профессиональным сообществом, имеет подробную документацию (см. https://scikit-learn.org/stable/user_guide.html), применяется в исследовательских, промышленных и образовательных проектах. Схема машинного обучения с использованием библиотеки Scikit-learn (Рисунок 13) содержит такие этапы как получение набора данных, очистка данных, конструирование признаков, формирование выборок, обучение на базе выбранного алгоритма, проверка и использование модели на новых данных.

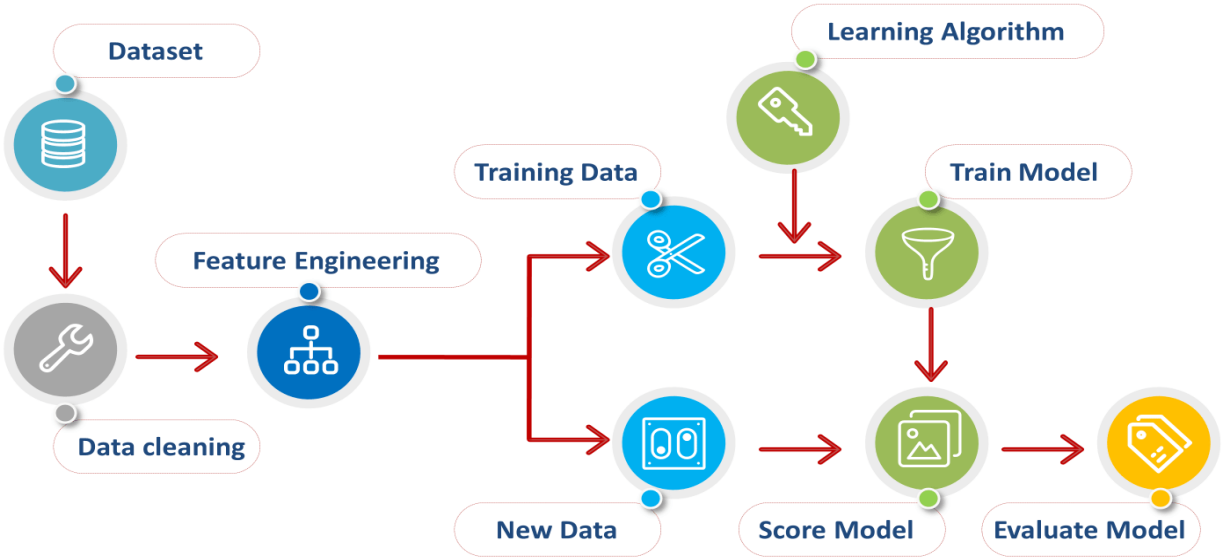


Рисунок 13 – Схема процесса машинного обучения

Среды реализуемых библиотекой методов находятся:

- **линейные:** модели, задача которых построить разделяющую (для классификации) или аппроксимирующую (для регрессии) гиперплоскость;
- **деревья решений:** обучение моделей, базирующихся на множестве условий, оптимально выбранных для решения задачи;
- **ансамблевые методы:** методы, основанные на деревьях решений, которые комбинируют множествj деревьев и повышают их качество работы, позволяют производить отбор признаков (бустинг, бэггинг, случайный лес);
- **нейронные сети:** комплексный нелинейный метод для задач регрессии и классификации;
- **SVM:** нелинейный метод, который обучается определять границы принятия решений;
- **наивный Байес**: прямое вероятностное моделирование для задач классификации;
- **PCA:** линейный метод понижения размерности и отбора признаков;
- **t-SNE:** нелинейный метод понижения размерности;
- **K-средних**: самый распространенный метод для кластеризации, требущий на вход число кластеров, по которым должны быть распределены данные;
- **кросс-валидация:** метод, при котором для обучения используется весь датасет (в отличие от разбиения на выборки train/test), однако обучение происходит многократно, и в качестве валидационной выборки на каждом шаге выступают разные части датасета;
- **Grid Search:** метод для нахождения оптимальных гиперпараметров модели.

Библиотеку можно использовать для решения широкого круга задач, для ориентации по методам библиотеки можно использовать приведённую ниже схему выбора метода (Рисунок 14).

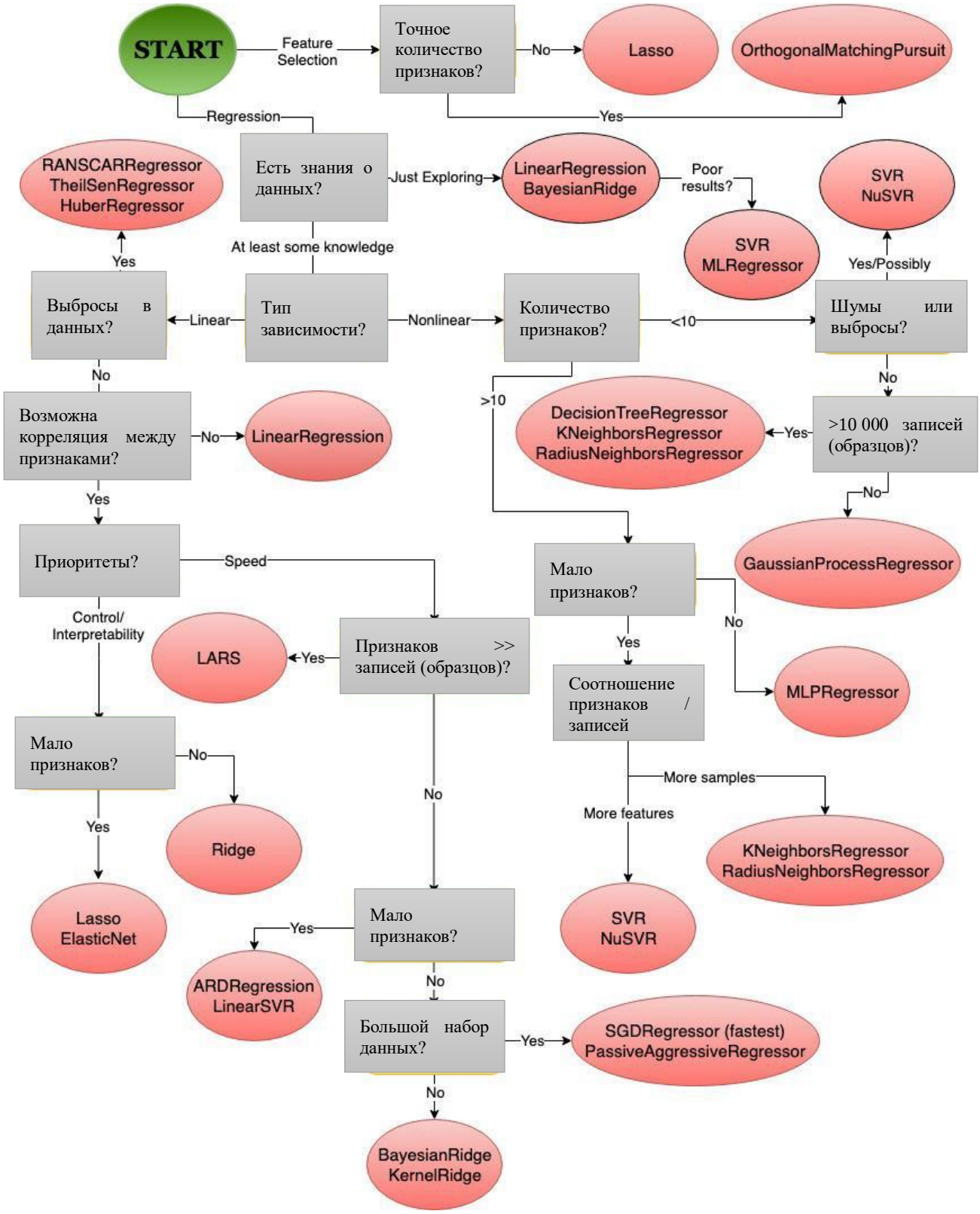


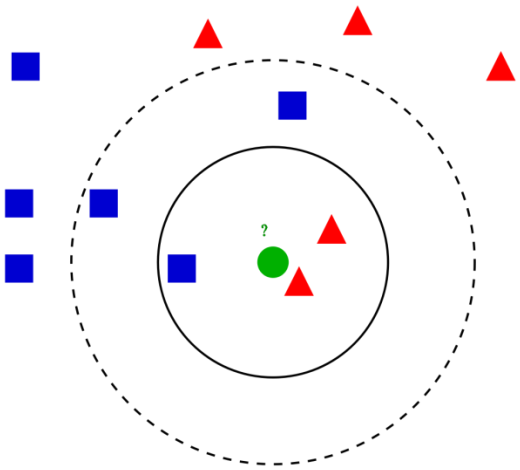
Рисунок 14 – Алгоритм выбора метода решения задачи машинного обучения

Типы классификаторов

Scikit-Learn даёт доступ ко множеству различных алгоритмов классификации. Вот основные из них:

- Метод k-ближайших соседей (K-Nearest Neighbors);
- Метод опорных векторов (Support Vector Machines);
- Классификатор дерева решений (Decision Tree Classifier) / Случайный лес (Random Forests);
- Наивный байесовский метод (Naive Bayes);
- Линейный дискриминантный анализ (Linear Discriminant Analysis);
- Логистическая регрессия (Logistic Regression);

Метод k-ближайших соседей (K-Nearest Neighbors)



Этот метод работает с помощью поиска кратчайшей дистанции между тестируемым объектом и ближайшими к нему классифицированным объектами из обучающего набора. Классифицируемый объект будет относится к тому классу, к которому принадлежит ближайший объект набора.

Классификатор дерева решений (Decision Tree Classifier)

Этот классификатор разбивает данные на всё меньшие и меньшие подмножества на основе разных критериев, т. е. у каждого подмножества своя сортирующая категория. С каждым разделением количество объектов определённого критерия уменьшается.

Классификация подойдёт к концу, когда сеть дойдёт до подмножества только с одним объектом. Если объединить несколько подобных деревьев решений, то получится так называемый *Случайный Лес* (англ. *Random Forest*).

Наивный байесовский классификатор (Naive Bayes)

Такой классификатор вычисляет вероятность принадлежности объекта к какому-то классу. Эта вероятность вычисляется из шанса, что какое-то событие произойдёт, с опорой на уже на произошедшие события.

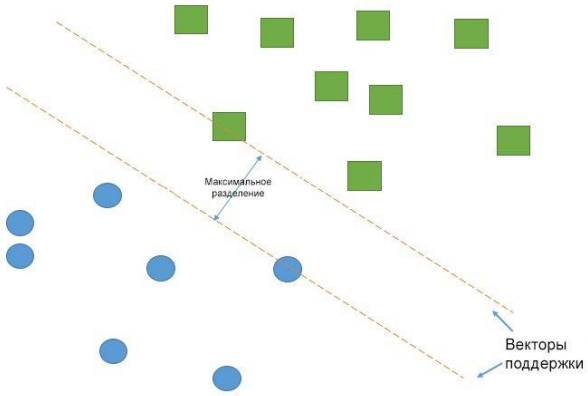
Каждый параметр классифицируемого объекта считается независимым от других параметров.

Линейный дискриминантный анализ (Linear Discriminant Analysis)

Этот метод работает путём уменьшения размерности набора данных, проецируя все точки данных на линию. Потом он комбинирует эти точки в классы, базируясь на их расстоянии от центральной точки.

Этот метод, как можно уже догадаться, относится к линейным алгоритмам классификации, т. е. он хорошо подходит для данных с линейной зависимостью.

Метод опорных векторов (Support Vector Machines)



Работа метода опорных векторов заключается в рисовании линии между разными кластерами точек, которые нужно сгруппировать в классы. С одной стороны линии будут точки, принадлежащие одному классу, с другой стороны — к другому классу.

Классификатор будет пытаться увеличить расстояние между рисуемыми линиями и точками на разных сторонах, чтобы увеличить свою «уверенность» определения класса. Когда все точки построены, сторона, на которую они падают — это класс, которому эти точки принадлежат.

Логистическая регрессия (Logistic Regression)

Логистическая регрессия выводит прогнозы о точках в бинарном масштабе — нулевом или единичном. Если значение чего-либо равно либо больше `0.5`, то объект классифицируется в большую сторону (к единице). Если значение меньше `0.5` — в меньшую (к нулю).

У каждого признака есть своя метка, равная только 0 или только 1. Логистическая регрессия является линейным классификатором и поэтому используется, когда в данных прослеживается какая-то линейная зависимость.

Примеры задач классификации

Задача классификации — эта любая задача, где нужно определить тип объекта из двух и более существующих классов. Такие задачи могут быть разными: определение, кошка на

изображении или собака, или определение качества вина на основе его кислотности и содержания алкоголя.

В зависимости от задачи классификации вы будете использовать разные типы классификаторов. Например, если классификация содержит какую-то бинарную логику, то к ней лучше всего подойдёт логистическая регрессия.

По мере накопления опыта вам будет проще выбирать подходящий тип классификатора. Однако хорошей практикой является реализация нескольких подходящих классификаторов и выбор наиболее оптимального и производительного.

Реализация классификатора

Первый шаг в реализации классификатора — его импорт в Python. Вот как это выглядит для логистической регрессии:

```
from sklearn.linear_model import LogisticRegression
```

Вот импорты остальных классификаторов, рассмотренных выше:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
```

Однако, это не все классификаторы, которые есть в Scikit-Learn. Про остальные можно прочитать на соответствующей странице в документации.

После этого нужно создать экземпляр классификатора. Сделать это можно создав переменную и вызвав функцию, связанную с классификатором.

```
logreg_clf = LogisticRegression()
```

Теперь классификатор нужно обучить. Перед этим нужно «подогнать» его под тренировочные данные.

Обучающие признаки и метки помещаются в классификатор через функцию `fit`:

```
logreg_clf.fit(features, labels)
```

После обучения модели данные уже можно подавать в классификатор. Это можно сделать через функцию классификатора `predict`, передав ей параметр (признак) для прогнозирования:

```
logreg_clf.predict(test_features)
```

Эти этапы (создание экземпляра, обучение и классификация) являются основными при работе с классификаторами в Scikit-Learn. Но эта библиотека может управлять не только классификаторами, но и самими данными. Чтобы разобраться в том, как данные и классификатор работают вместе над задачей классификации, нужно разобраться в процессах машинного обучения в целом.

Процесс машинного обучения

Процесс содержит в себе следующие этапы: подготовка данных, создание обучающих наборов, создание классификатора, обучение классификатора, составление прогнозов, оценка производительности классификатора и настройка параметров.

Во-первых, нужно подготовить набор данных для классификатора — преобразовать данные в корректную для классификации форму и обработать любые аномалии в этих данных. Отсутствие значений в данных либо любые другие отклонения — все их нужно обработать, иначе они могут негативно влиять на производительность классификатора. Этот этап называется предварительной обработкой данных (англ. *data preprocessing*).

Следующим шагом будет разделение данных на обучающие и тестовые наборы. Для этого в Scikit-Learn существует отличная функция traintestsplit.

Как уже было сказано выше, классификатор должен быть создан и обучен на тренировочном наборе данных. После этих шагов модель уже может делать прогнозы. Сравнивая показания классификатора с фактически известными данными, можно делать вывод о точности классификатора.

Вероятнее всего, вам нужно будет «корректировать» параметры классификатора, пока вы не достигните желаемой точности (т. к. маловероятно, что классификатор будет соответствовать всем вашим требованиям с первого же запуска).

Ниже будет представлен пример работы машинного обучения от обработки данных и до оценки.

Реализация образца классификации

```
# Импорт всех нужных библиотек
import pandas as pd
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

Поскольку набор данных iris достаточно распространён, в Scikit-Learn он уже присутствует, достаточно лишь заложить эту команду:

```
sklearn.datasets.load_iris
```

Тем не менее, тут ещё нужно подгрузить CSV-файл, который можно скачать здесь.

Этот файл нужно поместить в ту же папку, что и Python-файл. В библиотеке Pandas есть функция `read_csv()`, которая отлично работает с загрузкой данных.

```
data = pd.read_csv('iris.csv')

# Проверяем, всё ли правильно загрузилось

print(data.head(5))
```

Благодаря тому, что данные уже были подготовлены, долгой предварительной обработки они не требуют. Единственное, что может понадобиться — убрать ненужные столбцы (например `ID`) таким образом:

```
data.drop('Id', axis=1, inplace=True)
```

Теперь нужно определить признаки и метки. С библиотекой Pandas можно легко «нарезать» таблицу и выбрать определённые строки/столбцы с помощью функции `iloc()`:

```
# ".iloc" принимает row_indexer, column_indexer
X = data.iloc[:,:-1].values
# Теперь выделим нужный столбец
y = data['Species']
```

Код выше выбирает каждую строку и столбец, обрезав при этом последний столбец.

Выбрать признаки интересующего вас набора данных можно также передав в скобках заголовки столбцов:

```
# Альтернативный способ выбора нужных столбцов:
X = data.iloc['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm']
```

После того, как вы выбрали нужные признаки и метки, их можно разделить на тренировочные и тестовые наборы, используя функцию `train_test_split()`:

```
# test_size показывает, какой объем данных нужно выделить для тестового набора
# Random_state — просто сид для случайной генерации
# Этот параметр можно использовать для воссоздания определённого результата:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=27)
```

Чтобы убедиться в правильности обработки данных, используйте:

```
print(X_train)
print(y_train)
```

Теперь можно создавать экземпляр классификатора, например метод опорных векторов и метод k-ближайших соседей:

```
SVC_model = svm.SVC()
# В KNN-модели нужно указать параметр n_neighbors
# Это число точек, на которое будет смотреть
# классификатор, чтобы определить, к какому классу принадлежит новая точка
KNN_model = KNeighborsClassifier(n_neighbors=5)
```

Теперь нужно обучить эти два классификатора:

```
SVC_model.fit(X_train, y_train)
KNN_model.fit(X_train, y_train)
```

Эти команды обучили модели и теперь классификаторы могут делать прогнозы и сохранять результат в какую-либо переменную.

```
SVC_prediction = SVC_model.predict(X_test)
KNN_prediction = KNN_model.predict(X_test)
```

Теперь пришло время оценить точности классификатора. Существует несколько способов это сделать.

Нужно передать показания прогноза относительно фактически верных меток, значения которых были сохранены ранее.

```
# Оценка точности — простейший вариант оценки работы классификатора
print(accuracy_score(SVC_prediction, y_test))
print(accuracy_score(KNN_prediction, y_test))
# Но матрица неточности и отчёт о классификации дадут больше информации о производительности
print(confusion_matrix(SVC_prediction, y_test))
print(classification_report(KNN_prediction, y_test))
```

Вот, к примеру, результат полученных метрик:

```
SVC accuracy: 0.9333333333333333
KNN accuracy: 0.9666666666666667
```

Поначалу кажется, что KNN работает точнее. Вот матрица неточностей для SVC:

```
[[ 7  0  0]
 [ 0 10  1]
 [ 0  1 11]]
```

Количество правильных прогнозов идёт с верхнего левого угла в нижний правый. Вот для сравнения метрики классификации для KNN:

```
precision    recall  f1-score   support

Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.91      0.91      0.91        11
Iris-virginica       0.92      0.92      0.92        12

micro avg       0.93      0.93      0.93        30
macro avg       0.94      0.94      0.94        30
weighted avg       0.93      0.93      0.93        30
```


Когда дело доходит до оценки точности классификатора, есть несколько вариантов.

Точность классификации

Точность классификации измерять проще всего, и поэтому этот параметр чаще всего используется. Значение точности — это число правильных прогнозов, делённое на число всех прогнозов или, проще говоря, отношение правильных прогнозов ко всем.

Хоть этот показатель и может быстро дать вам явное представление о производительности классификатора, его лучше использовать, когда каждый класс имеет хотя бы примерно одинаковое количество примеров. Так как такое будет случаться редко, рекомендуется использовать другие показатели классификации.

Логарифмические потери

Значение Логарифмических Потерь (англ. *Logarithmic Loss*) — или просто *логлосс* — показывает, насколько классификатор «уверен» в своём прогнозе. Логлосс возвращает вероятность принадлежности объекта к тому или иному классу, суммируя их, чтобы дать общее представление об «уверенности» классификатора.

Этот показатель лежит в промежутке от 0 до 1 — «совсем не уверен» и «полностью уверен» соответственно. Логлосс сильно падает, когда классификатор сильно «уверен» в неправильном ответе.

Площадь ROC-кривой (AUC)

Такой показатель используется только при бинарной классификации. Площадь под ROC-кривой представляет способность классификатора различать подходящие и не подходящие какому-либо классу объекты.

Значение `1.0`: вся область, попадающая под кривую, представляет собой идеальный классификатор. Следовательно, `0.5` означает, что точность классификатора соответствует случайности. Кривая рассчитывается с учётом точности и специфичности модели. Подробнее о расчётах можно прочитать здесь.

Матрица неточностей

Матрица неточностей (англ. *Confusion Matrix*) — это таблица или диаграмма, показывающая точность прогнозирования классификатора в отношении двух и более классов. Прогнозы классификатора находятся на оси X, а результат (точность) — на оси Y.

Ячейки таблицы заполняются количеством прогнозов классификатора. Правильные прогнозы идут по диагонали от верхнего левого угла в нижний правый. Про это можно почитать в данной статье.

Отчёт о классификации

В библиотеке Scikit-Learn уже встроена возможность создавать отчёты о производительности классификатора. Эти отчёты дают интуитивно понятное представление о работе модели.

Тема 5. Задача регрессии. Линейная и нелинейная регрессия.

Методы для задачи регрессии в библиотеке Scikit-learn

Для решения задачи регрессии в библиотеке Scikit-learn реализовано несколько различных методов (Рисунок 15):

– линейные модели: линейная регрессия,

– деревья решений,

– машина опорных векторов,

– метод k-ближайшего соседа,

– ансамблевые методы.

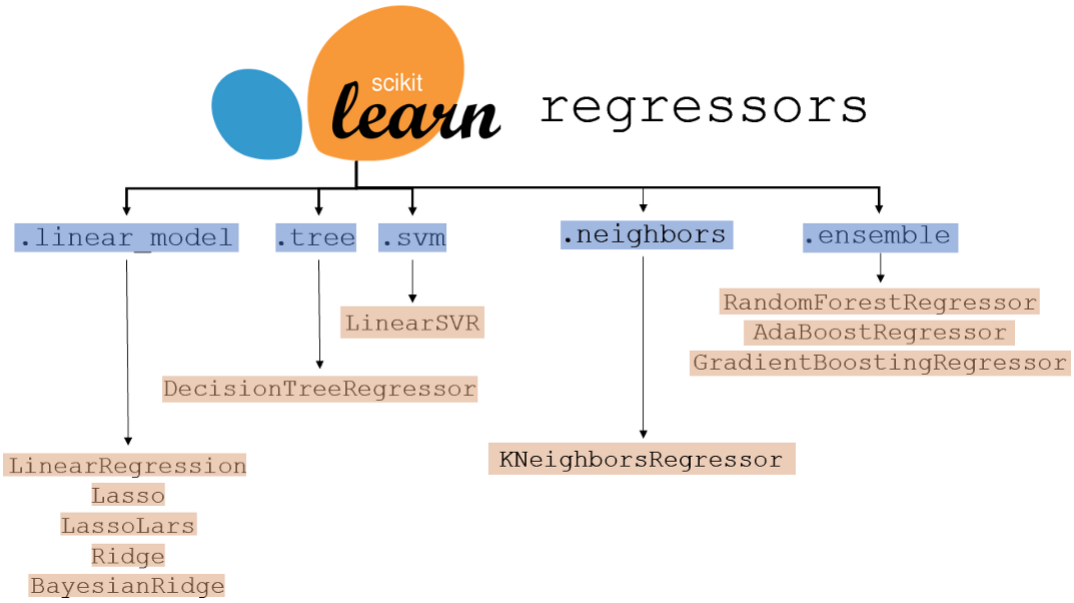


Рисунок 15 - Методы решения задачи регрессии в Scikit-learn [13]

Рассмотрим более подробно линейные модели для задачи регрессии. Все они находятся в модуле sklearn.linear_model.

Таблица 7 – Регрессионные методы в библиотеке Scikit-learn

Вид методов	Название метода в библиотеке	Описание
Классическая регрессия	linear_model.LinearRegression(*[, …])	Обычный метод наименьших квадратов для линейной регрессии
	linear_model.Ridge([alpha, fit_intercept, …])	Линейный метод наименьших квадратов с регуляризацией l2.
	linear_model.RidgeCV([alphas, …])	Гребневая регрессия со встроенной перекрестной проверкой
	linear_model.SGDRegressor([loss, penalty, …])	Линейная модель, подобранная путем минимизации регуляризованных эмпирических потерь с помощью SGD.
Регрессия с выбором	linear_model.ElasticNet([alpha, l1_ratio, …])	Линейная регрессия с объединенными априорными

[13] https://www.machinelearningmastery.ru/img/0-702902-390981.jpeg

переменных		значениями L1 и L2 в качестве регуляризатора.
	linear_model.ElasticNetCV(*[, l1_ratio, …])	Модель эластичной сети с итеративным подбором пути регуляризации
	linear_model.Lars(*[, fit_intercept, …])	Модель регрессии наименьшего угла
	linear_model.LarsCV(*[, fit_intercept, …])	Модель регрессии наименьшего угла с кросс-валидацией
	linear_model.Lasso([alpha, fit_intercept, …])	Линейная модель с L1-регуляризацией (также известного как Лассо).
	linear_model.LassoCV(*[, eps, n_alphas, …])	Линейная модель лассо с итеративной подгонкой по пути регуляризации.
	linear_model.LassoLars([alpha, …])	Модель лассо с подбором наименьшего угла
	linear_model.LassoLarsCV(*[, fit_intercept, …])	Модель Лассо с кросс-валидацией, использующая LARS алгоритм
	linear_model.LassoLarsIC([criterion, …])	Модель Лассо, использующая LARS алгоритм и BIC / AIC для модели выбора
	linear_model.OrthogonalMatchingPursuit(*[, …])	Модель ортогонального соответствия (OMP).
	linear_model.OrthogonalMatchingPursuitCV(*)	Модель ортогонального соответствия с кросс-валидацией
Байевская регрессия	linear_model.ARDRegression(*[, n_iter, tol, …])	Байевская ARD регрессия
	linear_model.BayesianRidge(*[, n_iter, tol, …])	Bayesian ridge regression.
Многозадачная линейная регрессия с выбором переменных	linear_model.MultiTaskElasticNet([alpha, …])	Многозадачная эластичная сеть с L1/L2 регуляризаций (mixed-norm)
	linear_model.MultiTaskElasticNetCV(*[, …])	Многозадачная эластичная сеть с L1/L2 регуляризаций (built-in) и с кросс-валидацией
	linear_model.MultiTaskLasso([alpha, …])	Многозадачная лассо модель с L1/L2 регуляризаций (mixed-norm)
Регрессия устойчивая к выбросам (робастные методы)	linear_model.HuberRegressor(*[, epsilon, …])	Модель линейной регрессии, устойчивая к выбросам.
	linear_model.QuantileRegressor(*[, …])	Модель линейной регрессии, которая прогнозирует условные квантили.
	linear_model.RANSACRegressor([…])	RANSAC (RANdom SAmple Consensus) алгоритм.
	linear_model.TheilSenRegressor(*[, …])	Theil-Sen Estimator: робастная многовариантная регрессионная модель
Обобщенные линейные модели регрессии	linear_model.PoissonRegressor(*[, alpha, …])	Обобщенная линейная модель с распределением Пуассона.
	linear_model.TweedieRegressor(*[, power, …])	Обобщенная линейная модель с распределением Твиди.
	linear_model.GammaRegressor(	Обобщенная линейная модель с

	*[, alpha, …])	гамма-распределением.

Более подробно о методах см. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model.

Для проверки адекватности получаемых моделей в библиотеке Scikit-learn используются разные метрики (Рисунок 16, наиболее часто используемые метрики отмечены зелёным).

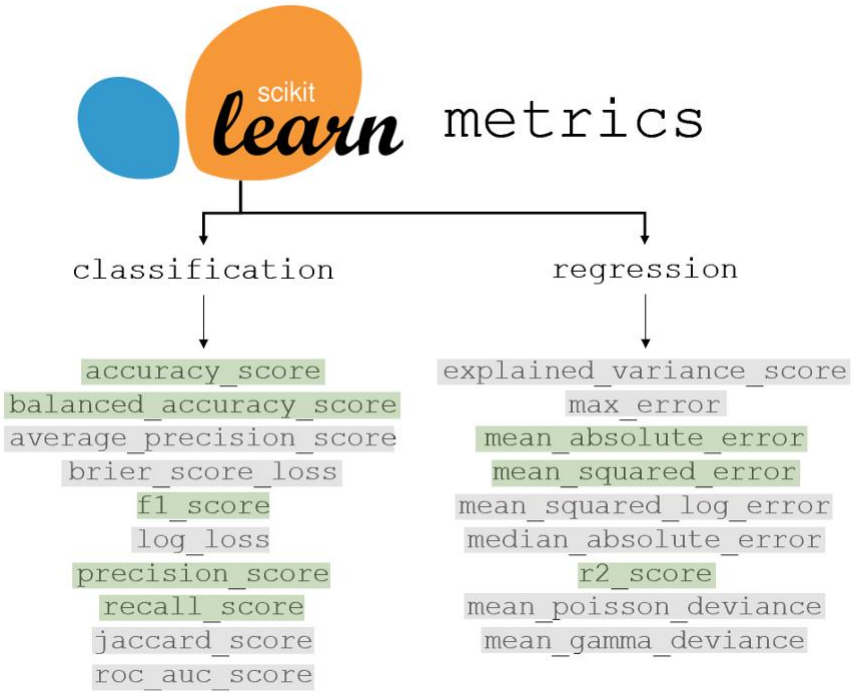


Рисунок 16 – Метрики для оценки моделей в задачах классификации и регрессии в библиотеке Scikit-learn[14]

Метрики реализованы в моделе metrics (Таблица 8).

Таблица 8 – Регрессионные метрики

metrics.explained_variance_score(y_true, …)	Объяснена функция оценки регрессии дисперсии.
metrics.max_error(y_true, y_pred)	max_error metric calculates the maximum residual error.
metrics.mean_absolute_error(y_true, y_pred, *)	Средняя абсолютная ошибка регрессии.
metrics.mean_squared_error(y_true, y_pred, *)	Среднеквадратичная ошибка регрессии.
metrics.mean_squared_log_error(y_true, y_pred, *)	Среднеквадратичная логарифмическая ошибка регрессии.
metrics.median_absolute_error(y_true, y_pred, *)	Медианная потеря регрессии абсолютной ошибки.
metrics.mean_absolute_percentage_error(…)	Средняя абсолютная процентная ошибка регрессии.

[14] https://www.kdnuggets.com/2021/01/ultimate-scikit-learn-machine-learning-cheatsheet.html

metrics.r2_score(y_true, y_pred, *[, …])	коэффициент детерминации
metrics.mean_poisson_deviance(y_true, y_pred, *)	Потеря регрессии среднего отклонения Пуассона.
metrics.mean_gamma_deviance(y_true, y_pred, *)	Потеря регрессии среднего гамма-отклонения.
metrics.mean_tweedie_deviance(y_true, y_pred, *)	Средняя потеря регрессии отклонения Твиди.

Более подробно о метриках см. https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics.

В библиотеке Scikit-learn все модели машинного обучения имеют функции:

– fit(x , y) для обучения модели на обучающей выборке (x,y),

– predict(x) для получения предсказанных значений на основе новых данных,

– set_params(**params) для установки параметров алгоритма обучения,

– get_params() для чтения параметров алгоритма обучения.

Кроме этого в зависимости от метода машинного обучения у модели имеют дополнительные методы (пример см.

Таблица 9 – Пример методов для разных моделей машинного обучения в библиотеке Scikit-learn

Методы класса	kNN	LDA	QDA	Logistic	SVC	Tree	RF	AdaBoost	GBT
fit(X, y)	+	+	+	+	+	+	+	+	+
predict(X)	+	+	+	+	+	+	+	+	+
predict_proba(X)	+	+	+	+		+	+	+	+
predict_log_proba(X)		+	+	+		+	+	+	+
score(X, y)	+	+	+	+	+	+	+	+	+
decision_function(X)		+	+	+	+			+	+
transform(X)		+		+		+	+		+
staged_decision_function(X)								+	+
staged_predict(X)								+	+
staged_predict_proba(X)								+	+
staged_score(X, y)								+	+
set_params(**params)	+	+	+	+	+	+	+	+	+
get_params()	+	+	+	+	+	+	+	+	+

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 10).

Таблица 10 – 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 kNN (k ближайших соседей)

При построении регрессионной модели на языке программирования Python можно использовать следующие библиотеки (Таблица 11):

Таблица 11 – Рекомендуемые библиотеки для построения регрессионной модели

Библиотека	**Роль при построении регрессионной модели**	**Описание**
NumPy	Хранение данных в массивах	это фундаментальный научный пакет Python, который позволяет выполнять множество высокопроизводительных операций с одномерными и многомерными массивами и реализует множество математических процедур.
Pandas	Хранение данных в виде датафреймов	программная библиотека на языке Python для обработки и анализа данных. Работа pandas с данными строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня.
Scikit-Learn	Реализация методов построения регрессионных моделей	библиотек машинного обучения, включает в себя различные алгоритмы классификации, регрессии и кластеризации и позволяет взаимодействовать с другими библиотеками численного моделирования, такими как Pandas, NumPy, Scipy.
StatsModels	Реализация методов построения регрессионных моделей с набором оценок модели	Библиотека, которая позволяет пользователям исследовать данные, оценивать статистические модели и выполнять статистические тесты.
Matplotlib	Визуализация данных на графиках	Библиотека для визуализации данных двумерной графикой.

Рассмотрим более подробно библиотеки для построения регрессионных моделей.

Построение линейной регрессионной модели с библиотекой Scikit-learn

Следуйте пяти шагам реализации линейной регрессии[15]:

1. Импортируйте необходимые пакеты и классы.
2. Предоставьте данные для работы и преобразования.
3. Создайте модель регрессии и приспособьте к существующим данным.
4. Проверьте результаты совмещения и удовлетворительность модели.

[15] https://proglib.io/p/linear-regression

5. Примените модель для прогнозов.

Это общие шаги для большинства подходов и реализаций регрессии.

Шаг 1: Импорт

Для импорта пакетом используем команду import. Для хранения данных нужды массивы, например, библиотеки NumPy, для построения линейной модели требуется класс LinearRegression из sklearn.linear_model:

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

Для реализации линейной модели могут использоваться и другие классы (см. Таблица 7).

Шаг 2: Данные

Регрессионная модель должна строится на основе данных, данные можно:

– сгенерировать (на базе математической функции или случайным образом),

– получить из базы данных (используя драйвер для взаимодействия с СУБД),

– загрузить из файла (например, используя библиотеку pandas).

Данных делятся на:

– входы (регрессоры, x),

– выход (предиктор, y).

Входы и выходы должны быть массивами (экземпляры класса numpy.ndarray) или похожими объектами. Например:

```
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])
```

Вызов .reshape() на **x необходим**, потому что этот массив должен быть **двумерным,** иметь **одну колонку и необходимое количество рядов**. Это как раз то, что определяет аргумент **(-1, 1)**.

Отличие реализации регрессионной модели множественной от парной в задании входных данных, факторы подаются в многомерном массиве.

Пример:

```
y = [1,2,3,4,3,4,5,3,5,5,4,5,4,5,4,5,6,0,6,3,1,3,1]
X = [[0,2,4,1,5,4,5,9,9,9,3,7,8,8,6,6,5,5,5,6,6,5,5],
     [4,1,2,3,4,5,6,7,5,8,7,8,7,8,7,8,6,8,9,2,1,5,6],
     [4,1,2,5,6,7,8,9,7,8,7,8,7,4,3,1,2,3,4,1,3,9,7]]
```

Множественная линейная регрессия - это модель, которая вычисляет отношение между двумя или более чем двумя переменными и одной переменной ответа путем подбора уравнения линейной регрессии между ними. Это помогает оценить зависимость или изменение зависимых переменных от изменения независимых переменных. В стандартной множественной линейной регрессии все независимые переменные учитываются одновременно.

Шаг 3: Построение модели

Для определения регрессионной модели нужно создать экземпляр выбранного класса линейной регрессии с соответствующими параметрами или оставить параметры по умолчанию, например:

```
model = LinearRegression()
```

Эта операция создаёт переменную model в качестве экземпляра LinearRegression. У класса LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False) можно использовать несколько параметров:

- **fit_intercept** – логический (True по умолчанию) параметр, который решает, вычислять отрезок b_0 (True) или рассматривать его как равный нулю (False), т.е следует ли вычислять точку пересечения (True) или считать ее равной нулю (False).
- **normalize** – логический (False по умолчанию) параметр, который решает, нормализовать входные переменные (True) или нет (False).
- **copy_X** – логический (True по умолчанию) параметр, который решает, копировать (True) или перезаписывать входные переменные (False).
- **n_jobs** – целое или None (по умолчанию), представляющее количество процессов, задействованных в параллельных вычислениях. None означает отсутствие процессов, при -1 используются все доступные процессоры.

Для обучения модели (вычисления коэффициентов на базе входов и выходов) нужно использовать метод `.fit()`:

```
model.fit(x, y)
```

Метод возвращает self - переменную model, поэтому можно заменить две последние операции на:

```
model = LinearRegression().fit(x, y)
```

Пример с другим классов `SGDRegressor` выполняет линейную регрессию с использованием градиентного спуска и принимает следующие аргументы:

- **tol** - сообщает модели, когда следует прекратить итерацию,
- **eta0** - начальная скорость обучения:

```
linear_regression_model = SGDRegressor(tol=.0001, eta0=.01)
linear_regression_model.fit(scaled_df, target)
predictions = linear_regression_model.predict(scaled_df)
```

Шаг 4: Получение результатов построения модели

Построенная модель должна быть оценена, для этого можно использовать коэффициент детерминации (R^2), его значение можно получить, используя метод `.score()`:

```
r_sq = model.score(x, y)
print('R2', r_sq)
```

Метод принимает в качестве аргументов предсказатель **x** и регрессор **y**, и возвращает значение R^2.

Для получения параметров регрессионной модели используются атрибуты .intercept (w0 – скалярное значение) и .coef (w1, w2.. – массив / вектор):

```
print('w0:', model.intercept_)
print('w1:', model.coef_)
```

Для получения оценок можно использовать sklearn.metrics. Для этого нужно выполнить команду import и расчитать нужные метрики (Таблица 8):

```
from sklearn.metrics import mean_squared_error,
mean_absolute_errornp.sqrt(mean_squared_error(y, yp))
mean_absolute_error(y, yp)
```

В данном примере, получаем оценки RMSE и MAE.

Если метрики указывают на недостаточную адекватность модели, но повысить их значение можно, используя:

– функции преобразования / масштабирования,
– обработка выбросов (если их много),
– добавление новых признаков,
– использование разных алгоритмов,
– настройка гиперпараметров модели.

Шаг 5: Использование модели

Построенную и верифицированную модель можно использовать для прогноза с текущими и новыми данными. Для этого используется метод.predict():

```
y_pred = model.predict(x)
print('прогнозируемое значение:', y_pred, sep='\n')
```

Метод реализует вычисления на основе входных данных. Линейная регрессионная парная модель может быть вычислена следующим образом:

```
y_pred = model.intercept_ + model.coef_ * x
print('прогнозируемое значение:', y_pred, sep='\n')
```

Вывод отличается от предыдущего примера количеством измерений. Теперь предсказанный ответ – это двумерный массив, в отличии от предыдущего случая, в котором он одномерный. Измените количество измерений **x** до одного, и увидите одинаковый результат.

Для получения прогноза на новых значениях, нужно на вход подать новый массив с данными:

```
x_new = np.arange(5).reshape((-1, 1))
print(x_new)
y_new = model.predict(x_new)
print(y_new)
```

Методические указания по использованию библиотеки StatsModels

Statsmodels – библиотека на Python, содержащая набор инструментов для статистического моделирования и эконометрии, включающий некоторую описательную статистику, оценку статистической модели и вывод.

Библиотека реализует несколько регрессионных методов (Таблица 12).

Таблица 12 – Регрессионные методы библиотеки

OLS(endog[, exog, missing, hasconst])	обычные наименьшие квадраты
WLS(endog, exog[, weights, missing, hasconst])	взвешенные наименьшие квадраты
GLS(endog, exog[, sigma, missing, hasconst])	обобщенный метод наименьших квадратов
GLSAR(endog[, exog, rho, missing, hasconst])	Generalized Least Squares with AR covariance structure
RecursiveLS(endog, exog[, constraints])	Рекурсивный метод наименьших квадратов
RollingOLS(endog, exog[, window, min_nobs, …])	Прокручивающиеся обыкновенные наименьшие квадраты
RollingWLS(endog, exog[, window, weights, …])	Покручивающиеся взвешенные наименьшие квадраты

Подробнее о методах см. https://www.statsmodels.org/stable/api.html.

Шаг 1: Импорт

Классы, реализующие различные регрессионные модели находятся в модуле statsmodels.regression.linear_model, но для интерактивного использования моделей рекомендуется подключать с помощью команды import библиотеку statsmodels.api. API

ориентирован на модели и наиболее часто используемые статистические тесты и инструменты. пример:

```
import statsmodels.api as sm
```

Такой импорт загрузит большую часть общедоступных частей statsmodels и делает большинство функций и классов доступными на одном или двух уровнях, не перегружая пространство имен «sm».

Другой вариант, импорт нужных моделей из statsmodels.regression.linear_model, например:

```
from statsmodels.regression.linear_model import OLS, WLS
```

Шаг 2: Данные

Данные определяются также, как и при использовании библиотеки Scikit-learn.

Шаг 3: Построение модели

При построении модели можно использовать один из методов библиотеки (Таблица 12), параметры их различные, но во всех присутствуют endog, exdog, которые представляют зависимую переменную и независимую переменную соответственно.

Рассмотрим метод statsmodels.OLS, который имееет 4 параметра:

– endog - зависимая переменная (statsmodels.OLS не предполагает, что модель регрессии имеет постоянный член, т.е. x0 (t) = 1k × (n + 1), в котором все значения в крайнем левом столбце равны 1, для обеспечения этого используют функцию sm.add_constant(), которая добавляет 1 слева от входного массива / списока / Series / DataFrame),
– exog - значение переменной регрессии, независимая переменная,
– missing – обработка пустых значений, принимает значения: ‘none’, ‘drop’, and ‘raise’. Если «none», проверка nan не выполняется, если «drop», все наблюдения с nans отбрасываются, если «raise», возникает ошибка; по умолчанию установлено ‘none’.
– hasconst – булева переменная, указывает, включается ли в обработку константа,если True, константа не проверяется и k_constant устанавливается в 1, если False, константа не проверяется и k_constant устанавливается в 0.

Пример вызова метода с установленными по умолчанию 3 и 4 параметром:

```
# строим регрессионную модель на базе метода "обычные наименьшие квадраты"
model = sm.OLS(y, x2)
```

Шаг 4: Получение результатов построения модели

Для вычисления параметров регрессионной модели необходимо запустити метод .fit(), результат будет получен в переменную results, которая является экземпляром класса statsmodels.regression.linear_model.RegressionResultsWrapper. Этот объект содержит информацию о построенной регрессионной модели.

Пример:

```
# получаем результаты построения модели (расчет параметров)
results = model.fit()
```

Переменная results относится к объекту, который содержит подробную информацию о результатах линейной регрессии. Для получения параметров регрессионной модели можно воспользоваться свойствами:

– params - рассчитанные коэффициенты регрессии b0, b1, ..., bn, которые выводятся массивом.

– rsquared – коэффициент детерминации,

– rsquared_adj скорректированный коэффициент детерминации.

Пример:

```
print("Параметры: ", results.params)
print("R2: ", results.rsquared)
```

Для получения сводной информации о регрессионной модели можно воспользоваться методом .summary, который выдаст сводную таблицу результатов (Рисунок 17, Таблица 13).

Пример:

```
# выводим статистические результаты по модели
print(results.summary())
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.908
Model:	OLS	Adj. R-squared:	0.899
Method:	Least Squares	F-statistic:	108.3
Date:	Fri, 18 May 2018	Prob (F-statistic):	4.96e-07
Time:	14:33:49	Log-Likelihood:	-16.538
No. Observations:	13	AIC:	37.08
Df Residuals:	11	BIC:	38.21
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0145	0.875	-0.017	0.987	-1.939	1.911
x1	0.7241	0.070	10.406	0.000	0.571	0.877

Omnibus:	0.989	Durbin-Watson:	1.738
Prob(Omnibus):	0.610	Jarque-Bera (JB):	0.829
Skew:	-0.394	Prob(JB):	0.661
Kurtosis:	2.046	Cond. No.	42.5

Рисунок 17 – Сводная результирующая таблица

Таблица 13 – Описание параметров сводной таблицы

Параметр	Название	Описание
No. Observarions	Размер выборки (номер наблюдения), n	
Df Residuals	Степень свободы остаточной суммы квадратов (остатков)	количество наблюдений -число параметров (модель Df + 1 (постоянные параметры) RSS = n-2
Df Model	Количество параметров модели (исключая постоянные параметры)	
R-squared	Коэффициент детерминации	указывает насколько тесной является связь между факторами регрессии и зависимой переменной, это соотношение объясненных сумм квадратов возмущений, к необъясненным. Чем ближе к 1, тем ярче выражена зависимость.
Adj. R-squared	Скорректированный(adjusted) коэффициент детерминации Тейла	Проблема с R-squared в том, что он по любому растет с числом факторов, поэтому высокое значение данного коэффициента может быть обманчивым, когда в модели присутствует множество факторов. Для того, чтобы изъять из *коэффициента корреляции* данное свойство был придуман скорректированный коэффициент детерминации.
F-statistic	Критерий Фишера, F	Используется для оценки значимости модели регрессии в целом, является соотношением объяснимой дисперсии, к необъяснимой. Если модель линейной регрессии построена удачно, то она объясняет значительную часть дисперсии, оставляя в знаменателе малую часть. Чем больше значение параметра — тем лучше.
Prob (F-statistic)	Вероятность	Уровень значимости (показатель F) подтверждает статистическую значимость величины R2
Log-likelihood	Значение функции правдоподобия подобранной модели	
AIC	Критерий Акаике (Akaike's information criterion)	Используется для сравнения моделей. Лучшая модель соответствует минимальному значению Абсолютное значение критерия не несет в себе полезной информации.
BIC	Байесовский информационный критерий (Bayesian information	критерий выбора модели из класса параметризованных моделей, зависящих от разного числа

	criterion)	параметров, аналог AIC с более строгой функцией штрафа.
t value	Критерий, основанный на t распределении Стьюдента.	Значение параметра в линейной регрессии указывает на значимость фактора, принято считать, что при t > 2 фактор является значимым для модели.
coef	Коэффициент	
std err	стандартные ошибки коэффициентов	
P>|t| (P-значение)	значения уровней значимости	
t	t-статистика	расчетные значения t-критерия (коэффициент деленный на стандартную ошибку)
[0.025 0.975]	Доверительный интервал для коэффициентов	Нижние 95% и Верхние 95% – нижние и верхние границы доверительных интервалов для коэффициентов регрессии
Omnibus	Комплексный тест Omnibus tests	используется для проверки значимости сразу нескольких параметров модели. Если мы отклоняем нулевую гипотезу комплексного теста, мы знаем, что по крайней мере один параметр модели является значимым. Если мы отвергаем нулевую гипотезу модели ANOVA, мы можем использовать апостериорные тесты, чтобы определить, какие средние по совокупности на самом деле различаются. Если мы отвергаем нулевую гипотезу модели множественной линейной регрессии, мы можем посмотреть на p-значения для отдельных коэффициентов модели, чтобы определить, какие из них являются статистически значимыми.
Prob（Omnibus）	Вероятность	
Skew	Коэффициент асимметрии	величина, характеризующая асимметрию распределения данной случайной величины (положителен, если правый хвост распределения длиннее левого, и отрицателен в противном случае, = 0 - распределение симметрично относительно математического ожидания)
Kurtosis	Коэффициент эксцесса (коэффициент островершинности)	мера остроты пика распределения случайной величины (для нормального распределения =0, положителен, если пик острый, отрицателен, если пик гладкий)

Durbin–Watson	статистика Дарбина-Уотсона	Проверяет автокорреляцию ($0 \leq d \leq 4$). Если автокорреляции нет, то значения критерия $d \approx 2$, при позитивной автокорреляции $d \approx 0$, при отрицательной — $d \approx 4$.
Jarque–Bera	Тест Харке—Бера	это статистический тест, проверяющий ошибки наблюдений на нормальность посредством сверки их третьего момента (асимметрия) и четвёртого момента (эксцесс) с моментами нормального распределения, у которого $S = 0$, $K = 3$. В тесте Харке—Бера проверяется нулевая гипотеза $H\,0 : S = 0$, $K = 3$ против гипотезы $H\,1 : S \neq 0$, $K \neq 3$ применим только при большом числе наблюдений, при малом следует использовать тест Шапиро –Уилка.
Prob(JB)		
Cond. No.		

Подробнее о результатах см. https://www.statsmodels.org/devel/generated/statsmodels.regression.linear_model.RegressionResults.html.

Описание показателей оценки для регрессионной оценки приведены ниже (Таблица 14).

Регрессионные модели могут быть оценены разными методами на:

– Коррелированность ошибок - тест Дарбина-Уотсона,
– Гетероскедастичность (дисперсия не постоянна) – тест Кука-Вайсберга, визуальный анализ,
– Мультиколлинеарность –VIF, главные компоненты,
– Нелинейность – RESET-тест Рамсея, визуальный анализ,
– Робастность (выбросы) – статистика Кука, форма распределений, D-статистика, визуальный анализ,
– Стохастичность регрессоров – тест Хаусмана.

Таблица 14 – Показатели для оценки регрессионных моделей

Обоз.	Название	Формула	Смысл оценки	Условия применения
RSS	Сумма квадратов остатков (необъяснённая) (Residual Sum of Squares)	$RSS = \sum_i (y_i - \hat{y}_i)^2$	измеряет необъяснённую часть дисперсии зависимой переменной	
ESS	Объяснённая регрессией сумма квадратов (Explained Sum of Squares)	$ESS = \sum_i (\hat{y}_i - \bar{y})^2$, где $\bar{y} = \frac{1}{n}\sum_i^n y_i$	измеряет объяснённую часть дисперсии зависимой переменной	
TSS	Общая сумма квадратов (Total Sum of Squares)	$TSS = RSS + ESS = \sum_i (y_i - \bar{y})^2$		
SEE	Стандартная ошибка регрессии	$SEE = \sqrt{\frac{RSS}{n-m-1}}$, где m – количество степеней свободы.	величина квадрата ошибки, приходящейся на одну степень свободы модели, чем меньше значение *SEE*, тем качественнее модель	
$r_{y,\hat{y}}$	Коэффициент корреляции Пирсона	$r_{y,\hat{y}} = \frac{\sum_i^n (y_i - \bar{y}) \cdot (\hat{y}_i - \bar{y})}{\sqrt{TSS \cdot ESS}}$	показывает силу линейной зависимости между двумя переменными. Он изменяется от -1 до 1. Близость к -1 говорит об отрицательной линейной зависимости, близость к 1 – о положительной.	Матрица корреляции не имеет смысла для модели нелинейной регрессии, поскольку она показывает только силу линейной зависимости.
R^2	Коэффициент детерминации	$R^2 = 1 - \frac{RSS}{TSS}$, $0 \le R^2 \le 1$, $R^2 = \frac{ESS}{TSS}$ – для модели с константой	показывает, какую часть общего рассеяния зависимой переменной объясняет независимая переменная, чем ближе значение коэффициента к 1, тем сильнее зависимость.	Для регрессионных моделей с одинаковым количеством признаков и свободным членом
R^2_{adj}	Скорректированный(adjusted) коэффициент	$R^2_{adj} = 1 - (1 - R^2)\frac{(n-1)}{(k-1)}$,		Для регрессионных моделей с разным количеством

	детерминации Тейла	$R^2_{adj} \leq R^2$, k –количество факторов (параметров), n- - количество наблюдений.	R^2_{adj} <= R^2	признаков и свободным членом
R^2_{ext}	Обобщённый (extended) коэффициент детерминации	$R^2_{ext} =$		
F	Критерий Фишера, F	$F = \frac{R^2}{1-R^2} \cdot (n-2)$	используется для проверки гипотезы о том, что коэффициенты регрессии одновременно равны нулю (проверяет гипотезу о том, что все факторы (кроме константы) являются незначимыми)	Сравнивается с табличными значениями
t	Критерий Стьюдента, t	случайная величина с нулевым математическим ожиданием (при выполнении нулевой гипотезы), а в знаменателе — выборочное стандартное отклонение этой случайной величины, получаемое как квадратный корень из несмещённой оценки дисперсии	предназначен для определения статистической значимости каждого коэффициента уравнения (проверяет гипотезу о незначимости одного коэффициента)	Сравнивается с табличными значениями
SSE	сумма квадратов, квадрат ошибки	$SSE = \sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2$	вычисляет сумму квадратов ошибок соответствующих точек подогнанных данных и исходных данных	Чем ближе SSE к 0, тем лучше выбор модели и ее подгонка, и тем успешнее прогнозирование данных.
MSE	среднеквадратическая ошибка	$MSE = SSE / n = \frac{1}{n} \sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2$	является средним значением суммы квадратов ошибок соответствующих точек между прогнозируемыми данными и исходными данными, то есть SSE / n	

RMSE		$RMSE = \sqrt{MSE} = \sqrt{SSE / n} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2}$	среднеквадратичное значение, стандартное отклонение	
DW	статистика Дарбина-Уотсона	$\approx 2 - 2\frac{\sum_{t=2}^{T} e_t e_{t-1}}{\sum_{t=1}^{T} e_t^2} = 2(1 - \rho_1),$ где — коэффициент автокорреляции первого порядка.	проверки остатков на автокорреляцию первого порядка (зависимость текущего значения от предыдущих). Её значение находится в промежутке от 0 до 4. В случае отсутствия автокорреляции *DW* близка к 2. Близость к 0 говорит о положительной автокорреляции, к 4 — об отрицательной.	неприменима к моделям без константы и к моделям, которые в качестве факторов используют лагированные значения объясняемой переменной. В этих случаях статистика может показывать отсутствие автокорреляции при её наличии.

Шаг 5: Использование модели

Построенную и верифицированную модель можно использовать для прогноза с текущими и новыми данными.

– для получения прогноза на используемых для построения входных данных .fittedvalues,

– для получения прогноза на новом массиве .predict().

Пример:

```
# используем модель для предстаказия на новых данных
x3 = sm.add_constant(x_new)
print('предсказание:', results.predict(x3), sep='\n')
```

Примеры построения регрессионной модели

1) Пример построения линейной регрессионной модели с использованием библиотеки Scikit-learn и Numpy

```
# подключаем библиотеку для построения графиков
import matplotlib.pyplot as plt
# подключаем библиотеку для работы с массивами
import numpy as np
# подключаем библиотеку для построения регрессионной модели
from sklearn.linear_model import LinearRegression
# подключаем модуль расчета метрик регрессионной модели
from sklearn.metrics import mean_squared_error, r2_score
# генерируем последовательность для значений факторной переменной, преобразуем ее в матрицу
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
# генерируем последовательности значений зависимой переменной
y = np.array([5, 20, 14, 32, 22, 38])
# проверяем размерность переменных
print( x.shape, y.shape)
# строим и сразу запускаем расчет для линейной регрессионной модели с параметрами по умолчанию
model = LinearRegression().fit(x, y)
# расчитываем предсказанные моделью значения определяемой переменной
#y_pred = model.intercept_ + model.coef_ * x
y_pred = model.predict(x)
print('предсказываемые значения:', y_pred, sep='\n')
# в sklearn не предусмотрена процедура определения статистической значимости регрессионных коэффициентов
# выводим оценки для регрессионной модели
```

```
#среднеквадратичная ошибка
rmse = mean_squared_error(y, y_pred)
#коэффициент детерминации
r2 = r2_score(y, y_pred)
# W1 | b1
# регрессионные коэффициенты от метода
print('Slope:' , model.coef_)
# Wo | bo
# свободный член от метода
print('Intercept:', model.intercept_)
print('Root mean squared error: ', rmse)
# качество модели (коэффициент  R2 )
print('R2 score: ', r2)
#print(model.score(x, y_pred))
# используем регрессионную модель для расчета с другими входными значениями
x_new = np.arange(5).reshape((-1, 1))
y_new = model.predict(x_new)
print(y_new)
# формируем график для вывода исходных точек и полученной регрессионной прямой
plt.scatter(x, y, s=10)
plt.xlabel('x')
plt.ylabel('y')
plt.plot(x, y_pred, color='r')
plt.show()
```

2) Пример построения линейной регрессионной модели с использованием библиотеки StatsModels

```
# подключаем библиотеку для построения регрессионной модели
import statsmodels.api as sm
# генерируем последовательность для значений факторной переменной, преобразуем ее в матрицу
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
# генерируем последовательности значений зависимой переменной
y = np.array([5, 20, 14, 32, 22, 38])
# задаем констанпу для случает пересечения???
x2 = sm.add_constant(x)
# строим регрессионную модель на базе метода "обычные наименьшие квадраты"
model = sm.OLS(y, x2)
# получаем результаты построения модели (расчет параметров)
results = model.fit()
# выводим статистические результаты по модели
print(results.summary())
```

```
print("Параметры: ", results.params)
print("R2: ", results.rsquared)
# используем модель для предстаказия на новых данных
x_new = np.arange(5).reshape((-1, 1))
x3 = sm.add_constant(x_new)
print('предсказание:', results.predict(x3), sep='\n')
#Строим график
pred_ols = results.get_prediction()
iv_l = pred_ols.summary_frame()["obs_ci_lower"]
iv_u = pred_ols.summary_frame()["obs_ci_upper"]
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(x, y, "o", label="data")
ax.plot(x, results.fittedvalues, "r--.", label="OLS")
#ax.plot(x, iv_u, "r--")
#ax.plot(x, iv_l, "r--")
ax.legend(loc="best")
plt.show()
```

3) **Пример построения линейной множественной регрессионной модели с использованием библиотеки StatsModels**

```
import statsmodels.api as sm
import numpy as np

y = [1,2,3,4,3,4,5,3,5,5,4,5,4,5,4,5,6,0,6,3,1,3,1]
X = [[0,2,4,1,5,4,5,9,9,9,3,7,8,8,6,6,5,5,5,6,6,5,5],
     [4,1,2,3,4,5,6,7,5,8,7,8,7,8,7,8,6,8,9,2,1,5,6],
     [4,1,2,5,6,7,8,9,7,8,7,8,7,4,3,1,2,3,4,1,3,9,7]]
def reg_m(y, x):
    ones = np.ones(len(x[0]))
    X = sm.add_constant(np.column_stack((x[0], ones)))
    for ele in x[1:]:
        X = sm.add_constant(np.column_stack((ele, X)))
    results = sm.OLS(y, X).fit()
    return results
print(reg_m(y, x).summary())
```

4) **Пример построения линейной регрессионной модели с использованием библиотеки Scikit-learn и Pandas16 с набором данных (см. объяснения по ссылке)**

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

[16] https://teletype.in/@hw_code/Regression-vs-Classification-v-chem-raznica-12-18

```
import numpy as np
import pandas as pd
# импортируем модель
from sklearn.linear_model
import LinearRegression from sklearn.cross_validation
import train_test_split
# импортируем библиотеку для вычисления метрики качества
from sklearn import metrics
data_path = http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv
# загружаем модель
data = pd.read_csv(data_path, index_col=0)
array_items = ['TV', 'radio', 'newspaper']
X = data[array_items]
y = data.sales
# разделяем выборку на тренировочную и валидационную
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
# создаем модель и обучаем ее
linearreg = LinearRegression() linearreg.fit(X_train, y_train)
# делаем предсказания y_predict = linearreg.predict(X_test)
# вычисляем RMSE (root-mean-squared-error, среднеквадратичная ошибка)
print(np.sqrt(metrics.mean_squared_error(y_test, y_predict)))
```

Тема 6. Задача кластеризации. Метод k-means.

Кластеризация — разбиение множества объектов на подмножества, называемые кластерами. Кластеризация, будучи математическим алгоритм имеет широкое применение во многих сферах: начиная с таких естественно научных областей как биология и физиология, и заканчивая маркетингом в социальных сетях и поисковой оптимизацией.

Существует множество алгоритмов кластеризации, однако ниже будет рассмотрен метод k-средних, так как он является наиболее лаконичным и простым для понимания.

Кластеризация методом k-средних:

Исходной задачей будет распределение произвольного количества n-мерных точек по k кластерам.

1. Случайным образом создаются k точек, в дальнейшем будем называть их центрами кластеров;
2. Для каждой точки ставится в соответствии ближайший к ней центр кластера;
3. Вычисляются средние арифметические точек, принадлежащих к определённому кластеру. Именно эти значения становятся новыми центрами кластеров;
4. Шаги 2 и 3 повторяются до тех пор, пока пересчёт центров кластеров будет приносить плоды. Как только высчитанные центры кластеров совпадут с предыдущими, алгоритм будет окончен.

Исходные данные алгоритма:

- n — количество строк;
- k — количество кластеров;
- dim — размерность точек (пространства).

Выходные данные алгоритма:

- cluster — двумерный массив размерностью dim * k, содержащий k точек — центры кластеров;
- cluster_content — массив, содержащий в себе k массивов — массивов точек принадлежащих соответствующему кластеру.

```
def clusterization(array, k):
 n = len(array)
 dim = len(array[0])

 cluster = [[0 for i in range(dim)] for q in range(k)]
 cluster_content = [[] for i in range(k)]

 for i in range(dim):
        for q in range(k):
                cluster[q][i] = random.randint(0, max_cluster_value)

 cluster_content = data_distribution(array, cluster)
```

Переменные заданы. Первичные центры кластеров созданы с помощью библиотеки random(стр. 9 - 11) max_claster_value — константа задающая примерные границы исходного множества;

При помощи функции data_ditribution() произведено первичное распределения точек по кластерам (стр. 13). Рассмотрим эту функцию подробнее:

```
def data_distribution(array, cluster):
 cluster_content = [[] for i in range(k)]

 for i in range(n):
        min_distance = float('inf')
        situable_cluster = -1
        for j in range(k):
                distance = 0
                for q in range(dim):
                        distance += (array[i][q]-cluster[j][q])**2

                distance = distance**(1/2)
                if distance < min_distance:
                        min_distance = distance
                        situable_cluster = j

        cluster_content[situable_cluster].append(array[i])

 return cluster_content
```

Для каждой строчки (стр. 5) высчитывается расстояние до каждого центра кластеров. Здесь применяется стандартный алгоритм:

1. За начальное кратчайшее расстояние (min_distance) берётся несоизмеримо большое со значениями точек число;
2. Затем происходит вычисление расстояния до центра каждого кластера;

$$p = \sqrt{\sum_{i=1}^{n} \left(x_n - x_n^{\text{центроид.}}\right)^2}$$

Вычисление расстояния между точкой и центром в n-мерном пространстве.

3. Если вычисленное расстояние меньше минимального, то минимальное расстояние приравнивается к вычисленному и точка привязывается к этому кластеру (situable_cluster);
4. После обработки точки, в массив cluster_content в выбранный кластер (situable_cluster) кластер вкладывается значение точки.

Функция возвращает массив cluster_content.

В дальнейшем, как и полагается, данная последовательность действий обращается в цикл:

```
privious_cluster = copy.deepcopy(cluster)
 while 1:
        cluster = cluster_update(cluster, cluster_content, dim)
        cluster_content = data_distribution(array, cluster)
        if cluster == privious_cluster:
                break
                privious_cluster = copy.deepcopy(cluster)
```

Данный цикл целостным образом описывает шаг 4 из описании алгоритм k-средних (см. выше).

После распределения точек по центрам кластеров происходит перераспределение уже центров кластеров по привязанным к ним точкам (стр. 2). Рассмотрим функцию cluster_content() подробнее:

```
def cluster_update(cluster, cluster_content, dim):
```

```
        k = len(cluster)
        for i in range(k): #по i кластерам
                for q in range(dim): #по q параметрам
                        updated_parameter = 0
                        for j in range(len(cluster_content[i])):
                                updated_parameter += cluster_content[i][j][q]
                        if len(cluster_content[i]) != 0:
                                updated_parameter    =    updated_parameter    /
len(cluster_content[i])
                        cluster[i][q] = updated_parameter
        return cluster
```

Для каждого кластера, для каждого из n измерений вычисляется новое значения с помощью незамысловатого среднего арифметического: стр. 8-9 — складываются все значения; стр. 11-12 — сумма делится на количество точек в кластере; стр. 13 — кластер принимает обновлённое значение.

На данном месте алгоритм заканчивает свою работу. Полный алгоритм выглядит следующим образом:

```
        def clusterization(array, k):
        n = len(array)
        dim = len(array[0])

        cluster = [[0 for i in range(dim)] for q in range(k)]
        cluster_content = [[] for i in range(k)]

        for i in range(dim):
                for q in range(k):
                        cluster[q][i] = random.randint(0, max_cluster_value)

        cluster_content = data_distribution(array, cluster)

        privious_cluster = copy.deepcopy(cluster)
        while 1:
                cluster = cluster_update(cluster, cluster_content, dim)
                cluster_content = data_distribution(array, cluster)
                if cluster == privious_cluster:
                        break
                privious_cluster = copy.deepcopy(cluster)
```

Перейдём к визуализации:

Визуализируем результат алгоритма для 3-х и 2-х мерного исходных пространств. Воспользуемся библиотекой mathplotlib:

```
        import matplotlib.pyplot as plt
        from mpl_toolkits import mplot3d
        import numpy as np
```

Визуализация для 2-х мерного пространства происходит следующим образом:

```
        def visualisation_2d(cluster_content):

        k = len(cluster_content)
        plt.grid()
        plt.xlabel("x")
        plt.ylabel("y")

        for i in range(k):
                x_coordinates = []
                y_coordinates = []
                for q in range(len(cluster_content[i])):
                        x_coordinates.append(cluster_content[i][q][0])
                        y_coordinates.append(cluster_content[i][q][1])
                plt.scatter(x_coordinates, y_coordinates)
        plt.show()
```

Grid() — создание сетки. xlabel(), ylabel() — названия осей. Затем в массивы, соответствующие осям вкладываются значения точек. После такой операции для каждого

кластера вызывается функция scatter() — разброс точек по плоскости. В конце вызывается функция отображения — show().

Аналогичным образом визуализируется результат алгоритма для 3-х мерного пространства:

```
def visualisation_3d(cluster_content):

ax = plt.axes(projection="3d")
plt.xlabel("x")
plt.ylabel("y")

k = len(cluster_content)

for i in range(k):
        x_coordinates = []
        y_coordinates = []
        z_coordinates = []
        for q in range(len(cluster_content[i])):
                x_coordinates.append(cluster_content[i][q][0])
                y_coordinates.append(cluster_content[i][q][1])
                z_coordinates.append(cluster_content[i][q][2])
        ax.scatter(x_coordinates, y_coordinates, z_coordinates)
plt.show()
```

Тема 7. Задача ассоциации.

Перед тем, как перейти к сути алгоритма, вам нужно определить 3 параметра:

1. Во-первых, нужно установить **размер** набора. Вы хотите определить двухэлементный, трёхэлементный набор или какой-нибудь еще?
2. Во-вторых, определить **поддержку** – это число транзакций, входящих в набор, разделенное на общее количество транзакций. Набор, который равен поддержке, является самым часто встречаемым набором.
3. В-третьих, определить **достоверность**, то есть условную вероятность определенного товара оказаться в корзине с другими товарами. Пример: чипсы в вашем наборе имеют 67%-ную вероятность оказаться в одной корзине с газировкой.

Простой алгоритм Apriori состоит из трех шагов:

1. **Объединение**. Просмотр базы данных и определение частоты вхождения отдельных товаров.
2. **Отсечение**. Те наборы, которые удовлетворяют поддержке и достоверности, переходят на следующую итерацию с двухкомпонентными наборами,
3. **Повторение**. Предыдущие два шага повторяются для каждой величины набора, пока не будет повторно получен ранее определенный размер.

Apriori обычно рассматривается как самообучающийся алгоритм, поэтому его часто применяют для поиска интересных шаблонов и отношений.

Существует модификация алгоритма Apriori, способная проводить классификацию маркированных данных

Пример:

```
# Загрузить данные
def loadDataSet():
    # 1, 2 и другие числа представляют продукт 1, продукт 2 и т. Д.
    # Дополнительный пример: ['теннисная ракетка', 'теннис', 'спортивная
обувь', 'бадминтон']
    goodList = [
        ['Теннисная ракетка', 'теннис', 'спортивная обувь'],
        ['Теннисная ракетка', 'теннис'],
        ['Теннисная ракетка'],
        ['спортивная обувь'],
        ['теннис', 'спортивная обувь', 'бадминтон'],
        ['Теннисная ракетка', 'теннис']
    ]
    # return [[1, 2, 3], [1, 2], [1], [3], [2, 3, 4], [1, 2]]
    return goodList


# Найдите частые наборы товаров
def createC1(dataSet):
    C1 = []
```

```
    for transaction in dataSet:
        for item in transaction:
            if not [item] in C1:
                C1.append([item])
    C1.sort()
    # frozenset () возвращает замороженный набор. После замораживания нельзя
больше добавлять или удалять элементы из набора.
    return list(map(frozenset, C1))

def scanD(D, CK, minSupport):
    '''
         ssCnt: количество вхождений каждого элемента / товара во всех
наборах данных - тип словаря {название товара: раз}
         : param D: Список продуктов без дублирования - список [продукт 1,
продукт 2]
         : param CK: Все наборы данных - двухуровневый список [{продукт 1,
продукт 2}, {продукт 1}]
         : param minПоддержка: минимальная поддержка
         : return: retList, supportData - это продукты (список), которые
больше или равны минимальной поддержке, и поддержка продуктов, которые
соответствуют минимальной поддержке (словарь)
    '''
    ssCnt = {}
    # Во всех наборах данных количество вхождений каждого элемента / товара
    for tid in D:
        for can in CK:
            if can.issubset(tid):
                if not can in ssCnt:
                    ssCnt[can] = 1
                else:
                    ssCnt[can] += 1

    # print(ssCnt)
    numItems = float(len(D))
    retList = []
    supportData = {}
    for key in ssCnt:
        support = ssCnt[key]/numItems
        if support >= minSupport:
            retList.insert(0, key)
        supportData[key] = support

    # print(retList)
    # print(supportData)
    return retList, supportData

# Частые элементы устанавливают попарную комбинацию
def aprioriGen(Lk, k):
    '''
         Часто используемые наборы элементов объединяются в пары, нам нужно
найти взаимосвязь между двумя товарами.
    :param Lk:
    :param k:
    :return:
    '''
    retList = []
    lenLk = len(Lk)
    for i in range(lenLk):
        for j in range(i+1, lenLk):
            L1 = list(Lk[i])[:k-2]
            L2 = list(Lk[j])[:k-2]
            L1.sort()
            L2.sort()
            if L1 == L2:
```

```
                retList.append(Lk[i] | Lk[j])
    # print(retList)
    return retList


def apriori(dataSet, minSupport=0.5):
    '''
         Введите алгоритм априори
         : param dataSet: collection
         : param minПоддержка: минимальная поддержка 0,5
    :return:
    '''
    C1 = createC1(dataSet)
    # print(C1)
    #  set  ()  Функция  создает  набор  неупорядоченных  и  неповторяющихся
элементов, которые можно использовать для проверки взаимосвязи, удаления
повторяющихся  данных,  а  также  для  вычисления  пересечения,  разницы,
объединения и т. д.
    D = list(map(set, dataSet))
    L1, supportData = scanD(D, C1, minSupport)
    L = [L1]
    # print(L)
    k = 2
    while(len(L[k-2])>0):
        CK = aprioriGen(L[k-2], k)
        Lk, supK = scanD(D, CK, minSupport)
        supportData.update(supK)
        L.append(Lk)
        k += 1

    return L, supportData


# Узнайте правила ассоциации
# Основная функция расчета правила
def generateRules(L, supportData, minConf=0.7):
    bigRuleList = []
    for i in range(1, len(L)):
        for freqSet in L[i]:
            H1 = [frozenset([item]) for item in freqSet]
            if(i > 1):
                rulesFromConseq(freqSet,   H1,   supportData,   bigRuleList,
minConf)
            else:
                calcConf(freqSet, H1, supportData, bigRuleList, minConf)
    return bigRuleList


def calcConf(freqSet, H, supportData, brl, minConf=0.7):
    '''

        : param freqSet: Набор данных, который необходимо проверить, не
превышает ли он минимального уровня достоверности.
        : param H: Измените содержимое списка freqSet на односимвольный
элемент
    :param supportData:
         : param brl: полный набор
         : param minConf: минимальная уверенность
    :return:
    '''
    prunedH=[]
    for conseq in H:
        conf = supportData[freqSet]/supportData[freqSet-conseq]
        if conf >= minConf:
```

```
            print(freqSet-conseq, '--->', conseq, 'conf:', conf)
            brl.append((freqSet-conseq, conseq, conf))
            prunedH.append(conseq)
    return prunedH

def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):
    m = len(H[0])
    if (len(freqSet) > (m+1)):
        Hmp1 = aprioriGen(H, m+1)
        Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)
        if(len(Hmp1) > 1):
            rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)


if __name__ == '__main__':
    # Загрузить данные
    dataSet = loadDataSet()
    # Найдите частые наборы товаров
    L, supportData = apriori(dataSet)
    # Узнайте правила ассоциации
    rules = generateRules(L, supportData, minConf=0.7)
```

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ

Список рекомендуемой литературы

Основная

1) Воронина, В. В. Теория и практика машинного обучения : учебное пособие / В. В. Воронина. — Ульяновск : УлГТУ, 2017. — 290 с. — ISBN 978-5-9795-1712-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/165053 Режим доступа: для авториз. пользователей.

2) Флах, П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных / Флах П. - Москва : ДМК Пресс, 2015. - 400 с. - ISBN 978-5-97060-273-7. - Текст : электронный // ЭБС "Консультант студента" : [сайт]. - URL : https://www.studentlibrary.ru/book/ISBN9785970602737.html - Режим доступа : по подписке.

дополнительная

3) Рашка, С. Python и машинное обучение : крайне необходимое пособие по новейшей предсказательной аналитике, обязательное для более глубокого понимания методологии машинного обучения / Рашка С. - Москва : ДМК Пресс, 2017. - 418 с. - ISBN 978-5-97060-409-0. - Текст : электронный // ЭБС "Консультант студента" : [сайт]. - URL : https://www.studentlibrary.ru/book/ISBN9785970604090.html - Режим доступа : по подписке.

4) Шарден, Б. Крупномасштабное машинное обучение вместе с Python / Шарден Б. , Массарон Л. , Боскетти А. , пер. с анг. А. В. Логунова. - Москва : ДМК Пресс, 2018. - 358 с. - ISBN 978-5-97060-506-6. - Текст : электронный // ЭБС "Консультант студента" : [сайт]. - URL: https://www.studentlibrary.ru/book/ISBN9785970605066.html - Режим доступа : по подписке.

учебно-методическая

5) Методические рекомендации для семинарских (практических) занятий, лабораторного практикума и самостоятельной работы по дисциплине «Машинное обучение» / составитель: С.В. Липатова - Ульяновск: УлГУ, 2022 –79 с.

Программное обеспечение

Anaconda (дистрибутив языков программирования Python и R), библиотеки (open source).