ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

МАШИННОЕ ОБУЧЕНИЕ

Основы программирования на PYTHON

Тимур Казанцев

12+

Тимур Казанцев

Искусственный интеллект и Машинное обучение. Основы программирования на Python

«ЛитРес: Самиздат»

2020

Казанцев Т.

Искусственный интеллект и Машинное обучение. Основы программирования на Python / Т. Казанцев — «ЛитРес: Самиздат», 2020

ISBN 978-5-532-04002-1

В этой книге мы расскажем вам об основных понятиях Искусственного интеллекта и Машинного обучения. Вы познакомитесь с основными алгоритмами и моделями, использующимися для решения абсолютно разных задач. Мы научимся предсказывать цены на квартиры, ВВП стран, распределим цветы на разные классы и даже построим собственную нейронную сеть, которая сможет предсказывать, что изображено на рисунке.Для желающих овладеть языком программирования Python, на котором решается большинство задач по машинному обучению, мы пройдем основы программирования на этом языке и научимся использовать его для построения моделей машинного и глубокого обучения.

ISBN 978-5-532-04002-1

Содержание

Введение 5
История развития Искусственного интеллекта 7
Различие между ИИ, машинным обучением, глубинным обучением и нейронными сетями 10
Примеры использования ИИ, МО и ГО 17
- Искусственный интеллект 18
- Машинное обучение 19
- Глубокое обучение 20

Основные задачи и методы машинного обучения 22
- Обучение с учителем и обучение без учителя 22
- Регрессия 28
- Классификация 32
- Кластеризация 40

Ансамблирование в машинном обучении 42
- Ансамбли 43
- Комитет большинства 44
- Бэггинг и бустинг 45
- Случайный лес 47

Будущее Искусственного интеллекта 50
Основы программирования на Python 53
- Для тех, кто знает основы Python 53
- Установка Python. Anaconda и обзор библиотек для МО 54
- Введение в Питон. Базовые команды. 57
- If – оператор 60
- Оператор While 63
- Строки 66
- Списки 68
- Словари 70

Построение моделей машинного обучения в Python 73
- Предсказание цен на квартиры с помощью метода линейной регрессии 73
- Предсказание ВВП в зависимости от цен на нефть 84
- Выжившие на Титанике. Модель классификации с помощью Метода опорных векторов 88
- Решение задачи Выживших на Титанике с помощью модели Дерева решений, Случайного леса и Бэггинга 98

Нейронные сети. Распознавание изображений 105
- Архитектура нейронной сети 109
- Создание модели нейронной сети 115
- Компиляция модели 116
- Обучение модели 117
- Предсказание изображений 119

Бонусная глава. Открытые датасэты для задач машинного обучения 122

Введение

В последнее время очень много разговоров об Искусственном интеллекте (Artificial Intelligence), Машинном обучении (Machine learning), Глубоком обучении (Deep learning), больших данных (Big Data), нейронных сетях (Neural networks) и многих других терминах и технологиях, о которых 30 лет назад можно было прочитать только в футуристических книгах. Однако это не с проста. Уже сегодня различные технологии искусственного интеллекта используются в нашей повседневной жизни.

Когда мы смотрим новостную ленту, новостные агрегаторы показывают нам именно те новости, которые нам могут быть наиболее интересны. То же самое происходит в социальных сетях, на Youtube, музыкальных сервисах, где нам показывают именно те видео, песни или изображения, которые скорее всего нам понравятся.

Компьютеры уже могут распознавать нашу речь и автоматический перевод от Яндекс и Google Translate работает на порядок лучше, чем всего лишь 5 лет назад.

Техники распознавания изображений и окружающей среды и местности применяются в автономных беспилотных автомобилях, которые уже ездят по миру, в том числе и в России. И количество автономных машин увеличивается огромными темпами.

Кроме того, ИИ используется банками при решении о выдаче кредита, отделами продаж и маркетинга в компаниях, чтобы предсказывать объемы продаж, и делать более персональные рекомендации для каждого клиента.

Огромные бюджеты тратятся на таргетированную рекламу, которая становится все более точечной благодаря технологиям машинного обучения.

Особо актуальным ИИ становится в медицине, где нейронные сети могут выявлять наличие серьезных заболеваний намного с большей точностью чем самые профессиональные доктора.

Как вы видите, спектр использования Искусственного интеллекта очень обширен и его технологии уже используются во многих сферах.

Так как ареал использования охватывает практически все области, это требует большого количества специалистов, разбирающихся в том, как работают алгоритмы искусственного интеллекта и машинного обучения. И именно поэтому сегодня любому, кто хочет развития своей карьеры, необходимо иметь как минимум базовое представление об ИИ и МО.

По различным оценкам, сейчас во всем мире примерно всего лишь 300 000 специалистов по искусственному интеллекту, и из них только 10 000 это очень сильные профессионалы, которые работают над масштабными проектами. Оценивается, что спрос в самое ближайшее время на таких специалистов вырастет до 30 миллионов человек и продолжит расти в дальнейшем. То есть налицо огромная нехватка экспертов, которые понимают и умеют работать с технологиями ИИ и МО.

Многие технологические гиганты, такие как Google, Яндекс, Netflix, Alibaba, Tencent, Facebook жалуются на нехватку высококлассных специалистов и не даром зарплаты для таких вакансий – одни из самых высоких на рынке.

Сегодня специалист с двумя – тремя годами опыта в области больших данных и ИИ может получать более 150 тысяч долларов в год в Америке, Европе и Китае, а лучшие специалисты зарабатывают от миллиона долларов в год и выше. Не стоит и говорить о многочисленных стартапах в области ИИ, которые запускаются каждую неделю и привлекают огромные раунды инвестиций.

Таким образом, если подытожить, то Искусственный интеллект уже используется вокруг нас многими компаниями и сервисами, порой даже когда мы этого не замечаем. В общем и

целом, он делает наш опыт взаимодействия с окружающей действительностью более персонализированным и удобным.

Есть масса областей и индустрий, где можно приложить на практике знания ИИ. И есть очевидная нехватка специалистов в этой области, и они будут востребованы в ближайшие пару десятилетий как минимум.

В этой книге мы дадим базовое представление о том, что такое Искусственный интеллект и машинное обучение, расскажем основные виды, алгоритмы и модели, покажем вам где искать данные для анализа, и попрактикуемся вместе с вами над решением некоторых реальных задач машинного обучения. После прочтения данной книги вы сможете общаться свободно на эти темы, и, если захотите, сможете в дальнейшем углубить свои знания в этой области с помощью более специализированных программ.

До встречи внутри книги!

История развития Искусственного интеллекта

Именно в последние несколько лет термины Искусственный интеллект, машинное обучение, нейронные сети, биг дата стали, пожалуй, одними из самых обсуждаемых тем во всем мире. Сегодня об искусственном интеллекте не говорит только ленивый. Однако, необходимо помнить, что ИИ – это не что-то новое, и этой дисциплине уже несколько десятков лет.

Задумываться о том, может ли у машин быть интеллект, начали еще в середине прошлого века. Еще в 1950 году английский математик Алан Тьюринг предложил Тест Тьюринга, цель которого заключалась в том, чтобы определить может ли машина мыслить и обмануть человека, заставив его поверить, что он общается с таким же человеком как и он сам, а не с компьютером.

В том же самом году фантаст Айзек Азимов ввел в терминологию Три закона робототехники, в котором указал, какими должны быть взаимоотношения между людьми и роботами.

Законы гласили:

Робот не может причинить вред человеку или своим бездействием допустить, чтобы человеку был причинён вред.

Робот должен повиноваться всем приказам, которые даёт человек, кроме тех случаев, когда эти приказы противоречат Первому Закону.

Робот должен заботиться о своей безопасности в той мере, в которой это не противоречит Первому или Второму Законам.

В 1955 году проходил семинар ученых, где обсуждали будущее компьютеров. Одним из присутствующих был Джон Маккарти, который первым ввел в обиход термин Искусственный интеллект. Поэтому именно 1955 год принято считать годом рождения ИИ. Через три года тот же Маккарти создал язык программирования Lisp, который стал основным в работе с ИИ на следующие несколько лет.

В 1956 году инженер Артур Сэмюэл создает первый в мире самообучающийся компьютер, который умеет играть в шашки. Шашки были выбраны из-за того, что в них присутствовали элементарные правила, и в то же время если вы хотели выиграть в них, то требовалось следовать определенной стратегии. Этот компьютер, созданный Сэмуэлэм, обучался на простых книжках по игре в шашки, в которых описывались сотни партий с хорошими и плохими ходами.

В этом же году Герберт Саймон, Алан Ньюэлл, и Клиффорд Шоу придумали программу, которая называлась Логический теоретик. Считается, что это одна из первых программ, обладающих ИИ. Логический теоретик хорошо справлялся с ограниченным кругом задач, например, задачи по геометрии, и даже смог доказать теорему о равностороннем треугольнике элегантнее, чем Бертран Рассел.

В следующем 1957 году Фрэнк Розенблатт придумал Перцептрон, который представлял собой обучаемую систему, действовавшую не только в соответствии с заданными алгоритмами и формулами, но и на основании прошлого опыта. Здесь важно отметить, что в перцептроне были впервые использованы нейронные сети. Уже тогда ученые понимали, что некоторые (нечеткие) задачи решаются человеком очень быстро, в то время как у компьютера они отнимают много времени. Поэтому подумали, что возможно необходимо воспроизвести структуру работы мозга человека, чтобы научить компьютер работать так же быстро. Поэтому простейшие элементы перцептрона назывались нейронами, потому что вели себя похожим образом, как и нейроны мозга человека. Компьютерная модель перцептрона была реализована в 1960 году в виде первого нейрокомпьютера, который был назван Марк-1.

А в 1959 году Массачусетский Технологический Институт основывает лабораторию Искусственного интеллекта.

Идем дальше в следующее десятилетие. Уже в 1961 году первый робот внедряется на производстве автомобилей компании General Motors.

В 1965 году был изобретен первый чат-бот Eliza. Eliza должна была имитировать психотерапевта, который расспрашивал у пациента о его состоянии и предлагал возможные решения или просто мог посочувствовать собеседнику. Оказалось, что в разговоре с Элайзой люди испытывали примерно такие же эмоции и чувства, как и при общении с настоящим человеком.

В 1974 году было изобретено первое беспилотное транспортное средство в лаборатории Стэнфордского университета, оно станет прототипом для следующих лунных модулей.

В 1978 году Дуглас Леннон – создал самообучающуюся систему Эвриско. Эта система не только уточняла уже известные закономерности, но и предлагало новые. Через несколько лет Эвриско научилась решать такие задачи как: моделирование биологической эволюции, очистка поверхности от химикатов, размещение элементов на интегральных схемах.

В 1989 году Карнеги Мэллон создает беспилотный автомобиль с использованием нейронных сетей.

В 1988 году компьютер Deep Thought играет против чемпиона мира по шахматам Гарри Каспарова, но проигрывает ему, через 8 лет у них проходит очередная игра, и опять Каспаров оказывается сильнее компьютера. Но уже всего лишь через год, в 1997 году – сильно апгрейденный шахматный суперкомпьютер Deep Blue от IBM одерживает победу над Гарри Гаспаровым, и он становится первым компьютером, выигравшим у действующего чемпиона мира по шахматам. Начиная с 2000-х годов компьютеры стабильно выигрывают у людей.

В 1999 году компания Sony анонсирует собачку Айбо, навыки и поведение которой развиваются со временем. В этом же году впервые Массачусетский технологический институт показывает эмоциональный ИИ под названием Кисмет, который может распознавать эмоции людей и реагировать на них соответствующим образом.

В 2002 году начинается массовое производство автономных пылесосов iRobot, которые умеют перемещаться по дому самостоятельно, избегая препятствий.

В 2009 году Google подключается к гонке компаний по разработке собственного беспилотного автомобиля.

В 2011 году появляются Siri, Google Now и Cortana, умные виртуальные ассистенты. В 2014 году к ним присоединится Alexa от Amazon, а в 2017 году Алиса от Яндекса.

Помните мы говорили про Тест Тьюринга, который был изобретен Аланом Тьюрингом в 1950 году и был предназначен чтобы понять сможет ли ИИ обмануть человека и убедить его, что перед ним не компьютер, а человек. Так вот в 2014 году компьютерный чатбот Эжен Густман прошел этот тест, заставив треть жюри поверить, что компьютером управлял человек, а не ИИ.

В 2016 году Deep Mind от Google под названием Alpha Go побеждает чемпиона по игре в Го. Игра Го намного сложнее шахмат, здесь больше вариантов развития игры, и тем не менее Го стала второй игрой, в которой люди больше не могут выиграть.

В 2017 году после более чем 10 лет попыток и неудач, две команды независимо друг от друга разработали свои модели ИИ, компьютеры DeepStack и Libratus, которые смогли обыграть профессионалов в покер. В отличие от Го и шахмат, где все подчиняется строгим правилам, в покере на первый план выходит человеческий фактор. Потому что покер – во многом психологическая игра, построенная на эмоциях, невербальной коммуникации, умении блефовать и распознавать блеф.

Один из участников игры в покер с этими компьютерами так описал свои впечатления: «Это как играть с кем-то, кто видит все твои карты. Я не обвиняю нейросеть в нечестной игре, просто она действительно настолько хороша».

В 2015 году Илон Маск и Сэм Альтман, президент Y Combinator, основали компанию OpenAI, чтобы создать «открытый и дружественный» искусственный интеллект.

В 2017 году команда разработчиков OpenAI решила натренировать свою нейросеть в крупнейшей киберспортивной игре Dota 2. В этой игре играют команды по 5 человек, и они используют множество комбинаций из более чем сотни героев. У каждого из них есть свой набор навыков, За две недели нейросеть смогла обучиться и победить нескольких лучших игроков мира в режиме один на один, и сейчас ее создатели готовятся выпустить версию для основного режима, пять на пять.

Перемещаемся еще ближе к нашим дням. В начале 2018 году алгоритмы от Alibaba и Microsoft превзошли человека в тесте на понимание прочитанного текста.

В марте 2018 года небольшой робот собрал кубик Рубика за 0,38 секунды. Рекорд среди людей до этого составлял – 4,69 секунды.

Одним из самых важных прорывов в развитии ИИ, который может принести много пользы человечеству, стало то, что в мае 2018 искусственный интеллект стал лучше людей распознавать рак кожи.

Кроме распознавания заболеваний у пациентов, алгоритмы ИИ используются сегодня для исследования сворачивания белка, пытаясь найти лекарство от болезней Альцгеймера и Паркинсона. ИИ используется также для снижения уровня потребляемой энергии, и создания новых революционных материалов.

Искусственный интеллект активно применяется и в бизнесе – банки используют его для одобрения кредитов, а розничные компании применяют его для более точечных рекламных компаний и предложений для своих клиентов.

Почему же именно в наше время ИИ стал так быстро набирать скорость. Этому есть две причины. Во-первых, сейчас в мире производится огромное количество информации. Каждые два года, объем информации в мире удваивается. А как мы знаем, ИИ учится на имеющихся данных. И вторая причина – это наличие сильных вычислительных мощностей. Наши компьютеры сегодня достаточно сильные, чтобы они умели обрабатывать эти объемы информации в достаточно ограниченные сроки.

Итак, мы посмотрели на краткую историю развития ИИ. В одной из следующих глав мы посмотрим, чего же можно ожидать от развития ИИ в будущем.

Различие между ИИ, машинным обучением, глубинным обучением и нейронными сетями

Сегодня зачастую термины искусственный интеллект, машинное обучение (МО), глубокое обучение (ГО), нейронные сети (НС), Биг Дата используются взаимозаменяемо. И хотя они действительно очень связаны между собой, давайте разберемся что представляет собой каждое из этих понятий, и чем они отличаются.

Во-первых, если говорить очень кратко, то искусственный интеллект – это достаточно широкая отрасль, которая в свою очередь охватывает и машинное и глубокое обучение. МО является подвидом ИИ, а ГО является подвидом МО.

Под ИИ подразумевается, что компьютер может выполнять такие задачи, которые может выполнять и человек. И здесь дело касается не просто каких-то механических действий, например, поднять и отнести какой-то предмет, а задачи, которые требуют интеллектуального мышления, то есть, когда надо принять правильное решение. Например, задача выиграть в шахматы, или распознать что изображено на картинке, или понять, что было произнесено собеседником и выдать правильный ответ.

Для этого компьютеру дают множество правил или алгоритмов, следуя которым, он смог бы поступать так же, как поступал бы человек.

ИИ может быть узким (narrow AI) либо его еще иногда называют слабым, то есть когда машина может справляться только с ограниченным видом задач, лучше чем человек. Например, распознать, что на картинке или сыграть в шахматы и выиграть. Именно на этом этапе развития ИИ мы сейчас находимся. Следующий этап – это общий ИИ (general AI), когда ИИ может решить любую интеллектуальную задачу так же хорошо, как человек. И финальный этап – это сильный ИИ, когда ИИ справляется с большинством задач намного лучше, чем человек.

Как мы уже сказали, ИИ – это достаточно обширная область знаний. Она включает в себя следующие направления.

1. Обработка естественного языка, когда компьютер должен понимать, что написано, и выдать правильный и релевантный ответ. Сюда же входят переводы текстов и даже составление сложных текстов компьютерами.

2. Экспертные системы – это компьютерные системы, которые имитируют способность принятия решений человеком, в основном с помощью правил «если – то», нежели с использованием какого-то кода.

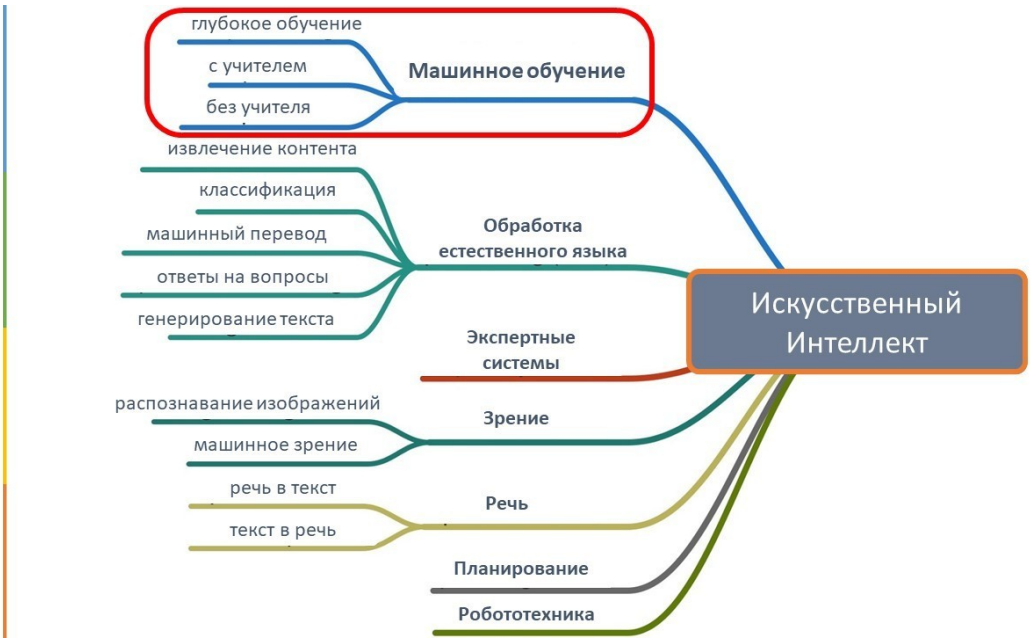
3. Речь – компьютер должен распознавать речь людей и сам уметь разговаривать.

4. Компьютерное зрение – компьютеры распознают те или иные объекты на изображении или при движении.

5. Робототехника – также очень популярное направление ИИ, создание роботов, которые могут выполнять различные функции, в том числе двигаться и общаться, преодолевать препятствия.

6. Автоматическое планирование – обычно используется автономными роботами и беспилотными аппаратами, когда им необходимо выполнять последовательность действий, особенно когда это происходит в многомерном пространстве и когда им приходится решать комплексные задачи.

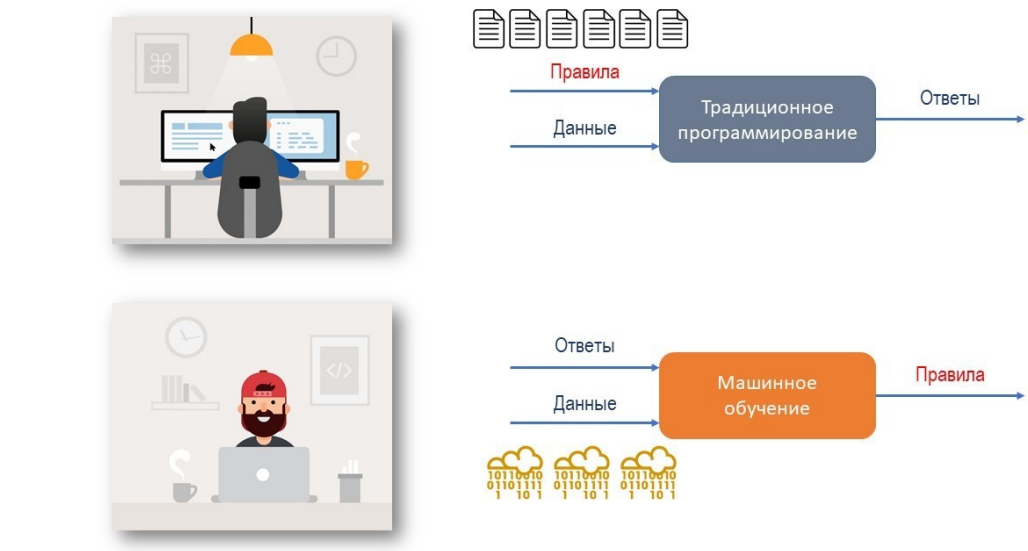
7. И наконец, Машинное обучение.



Машинное обучение появилось после того, как долгое время мы пытались сделать компьютер умнее, давая ему все больше и больше правил и инструкций. Однако, это оказалось не такой уж и хорошей идеей, потому что отнимало много времени, и мы не могли придумать правила для каждой детали и для каждой ситуации.

И тогда ученые пришли к выводу, а почему бы не написать алгоритмы, которые учатся самостоятельно на основе опыта. Так родилось машинное обучение. То есть, когда машины могут учиться на основе больших наборов данных вместо явно написанных инструкций и правил.

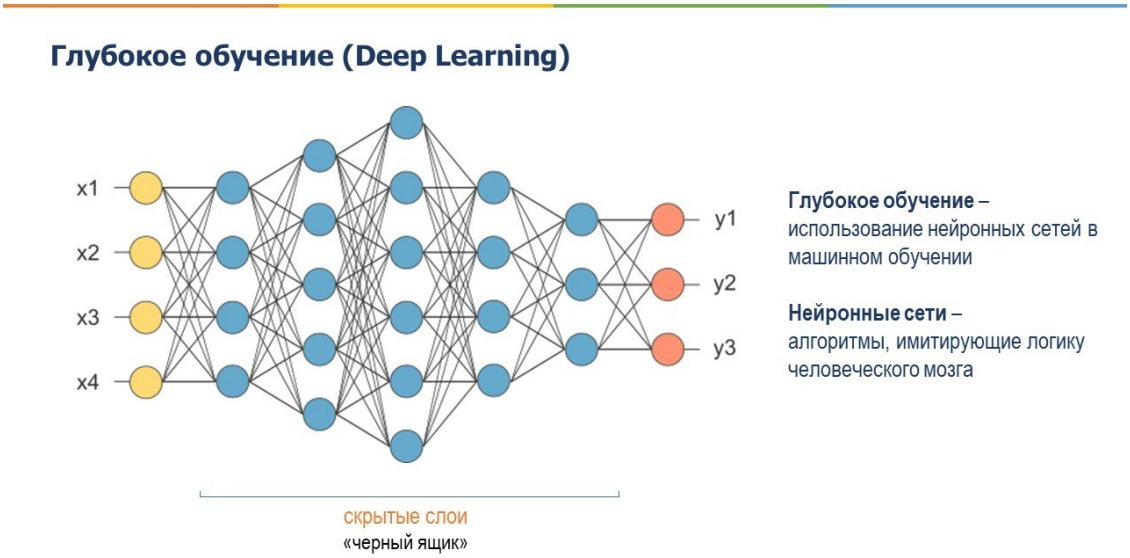
МО – это область ИИ, когда мы тренируем наш алгоритм с помощью набора данных, делая его все лучше, точнее и более эффективным. При машинном обучении наши алгоритмы обучаются на основе данных, но без заранее запрограммированных инструкций. То есть мы даем машине большой набор данных, и говорим правильные ответы, и потом машина сама создает алгоритмы, которые бы удовлетворяли этим ответам. И с каждым новым дополнительным объемом данных, машина учится дальше и еще больше улучшает свою точность прогнозов.



Например, если взять пример шахмат, то в примере с ИИ, мы даем машине много логических правил, и на их основе она учится играть. А в примере с МО, мы даем машине много примеров прошлых игр, она изучает их и анализирует почему одни игроки выигрывали, а другие проигрывали, какие шаги вели к успеху, а какие – к поражению. И на основе этих примеров, машина сама создает алгоритмы и правила как надо играть в шахматы, чтобы выиграть.

Другой пример, предположим, нам надо понять, как будет вести себя цена квартиры при изменении тех или иных параметров, например, в зависимости от площади, удаленности от метро, этажности и прочих факторов. Мы загружаем данные с разными квартирами, и компьютер создает модель, по которой можно будет предсказать цены в зависимости от этих факторов. Мы можем регулярно обновлять эти данные, и наш алгоритм будет обучаться на основе этих новых данных и каждый раз будет усовершенствовать свою точность по предсказанию цены в зависимости от параметров.

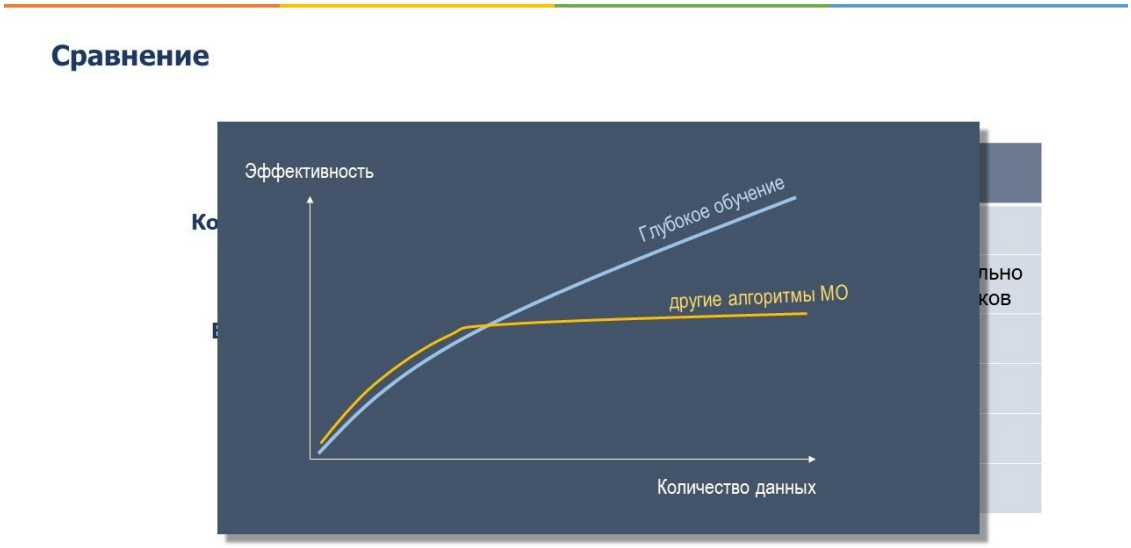
Идем дальше. Глубокое обучение – это подотрасль МО, то есть здесь тоже компьютер обучается, но обучается немного по-другому, чем в стандартном МО. В ГО используются нейронные сети (НС), которые представляют собой алгоритмы, повторяющие логику нейронов человеческого мозга. Большие объемы данных проходят через эти нейронные сети, и на выходе выдаются уже готовые ответы. Нейронные сети намного сложнее, чем обычное машинное обучение, и мы можем не всегда понимать, какие факторы имеют больший вес на тот или иной ответ, но использование нейронных сетей также помогает решать очень запутанные задачи в наше время. Иногда нейронные сети называют даже черным ящиком, потому что мы не всегда можем понять, что происходит внутри этих сетей.



Предположим, ваш компьютер оценивает, насколько хорошо написано эссе. Если вы используете ГО, то компьютер вам просто выдаст финальное решение, что эссе хорошее либо нет, и скорее всего, ответ будет очень близок к тому, как бы оценил это эссе человек. Но вы не сможете понять, почему было принято такое решение, потому что в ГО используются несколько уровней НС, что делает его очень трудно интерпретируемым. Вы не будете знать какой узел НС был активирован, и как эти узлы вели себя вместе, чтобы прийти к этому результату. Если же вы используете МО, например, алгоритм «дерево решений», то там видно какой фактор сыграл решающую роль в определении качества эссе.

Нейронные сети были известны еще в 20 веке, но тогда они были не настолько глубокими, там был всего один или два слоя, и они не давали таких хороших результатов, как другие алгоритмы МО. Поэтому на какое-то время они отошли на второй план. Однако они стали популярны в последнее время, особенно примерно с 2006 года, когда появились огромные наборы данных и сильные компьютерные мощности, в частности, видео карты и мощные процессоры, которые стали способны создавать более глубокие слои НС и делать вычисления более эффективно.

По этим же причинам, ГО является достаточно дорогим. Потому что, во-первых, сложно собрать большие данные по определенным признакам и, во-вторых, серьезные вычислительные способности компьютеров – тоже достаточно дорогое удовольствие.



Если вкратце, то каким образом работает ГО. Предположим, наша задача вычислить сколько единиц транспорта и какой именно транспорт (то есть автобусы, грузовики, машины или велосипеды) проходит через определенную трассу в день, чтобы в дальнейшем распределить полосы движения.

Для этой цели нам надо научить наш компьютер распознавать виды транспорта. Если бы мы решали эту задачу с помощью МО, мы бы написали алгоритм, в котором указывали бы характеристики машин, автобусов, грузовиков и велосипедов, например, если количество колес 2, то мотоцикл, если длина движущегося средства более 5 метров, то грузовик либо автобус, если много окон, то автобус, и т.д. Но как понимаете, здесь много подводных камней. Например, автобус может быть затонированным и будет трудно понять, где там окна, либо грузовик может выглядеть как автобус или наоборот, да и крупные машины пикапы выглядят как некоторые небольшие грузовики.

Поэтому другой вариант решения этой задачи, это загрузить большое количество изображений с разными видами транспорта в наш компьютер и просто указать ему, на каких изображениях изображен мотоцикл, автомобиль, грузовик или автобус. Компьютер сам начнет подбирать характеристики, по которым можно определить, что за вид транспорта изображен и как их можно отличить друг от друга. После этого мы загрузим еще некоторое количество изображений и протестируем насколько хорошо компьютер справляется с задачей. Если он будет ошибаться, мы укажем ему, что вот здесь ты ошибся, здесь не грузовик, а автобус. Компьютер, в свою очередь, вернется назад к своим алгоритмам (это называется backpropagation) и внесет туда какие-то изменения, и мы начнем заново по кругу до тех пор, пока компьютер не начнет угадывать, что изображено на картинке с очень большой долей вероятности. Это и называется глубокое обучение на основе нейронных сетей. Как вы понимаете, это может занимать достаточно долгое время, может быть несколько недель, в зависимости от сложности поставленной задачи, также требует наличия большого количества данных, желательно, чтобы было от миллиона изображений и выше, и все эти изображения должны либо быть промаркированы, либо это должен делать человек, но это будет очень затратно по времени.

Давайте еще раз сравним МО и ГО по разным параметрам.

Если суммировать:

ГО является подобластью МО, и они оба подпадают под более широкое определение ИИ.

МО использует алгоритмы, чтобы разбирать данные, обучаться на их основе, и принимать взвешенные решения на основе обученного.

ГО делает то же самое, так как оно тоже является разновидностью МО, но специфика ГО в том, что при нем алгоритмы структурируются в несколько слоев, чтобы создать искусственную нейронную сеть, которая может тоже обучаться и принимать умные решения.

МО может использоваться при небольших наборах данных. И на маленьких объемах данных, МО и ГО имеют примерно одинаковую эффективность, но при возрастании объемов данных, ГО намного выигрывает по эффективности.

В МО мы сами задаем характеристики, на которые будут опираться наши алгоритмы. В примере с определением цены квартиры, мы сами указываем параметры, от которых будет зависеть цена, например, метраж, расстояние от метро, возраст дома, район и т.д. А в ГО, компьютер или можно сказать нейронная сеть сама методом проб и ошибок выводит определенные параметры и их вес, от которых зависят наши выходные данные.

По времени обучения алгоритмов, ГО как правило занимает больше времени чем МО.

Расшифровка или интерпретация алгоритмов МО легче, потому что мы видим какой параметр играет важную роль для определения выходных данных. Например, в вопросе определения цены квартиры, мы можем увидеть, что вес метража в цене составляет, скажем, 60%. В ГО же, расшифровать что именно привело к такому результату порой бывает очень сложно, потому что там несколько слоев нейронных сетей и много параметров, которые компьютер выводит сам и которые он может посчитать важными. Поэтому, использование ГО или МО будет также зависеть от целей ваших задач. Например, если вам надо понимать, почему компьютер принял то или иное решение, какой фактор сыграл важную роль, то вам надо будет выбрать использование МО вместо ГО.

Вследствие того, что ГО требует большего объема данных, а также более мощных вычислительных способностей компьютера, и занимает больше времени для обучения, оно также является более дорогим по сравнению с МО.

Сравнение

	Машинное обучение	Глубокое обучение
Количество данных	небольшое	очень большое
Признаки	необходимо предоставить / создать самому	обучается самостоятельно без указания признаков
Время обучения	коротко	долго
Точность	хорошая	очень высокая
Расшифровка	легко	очень сложно
Стоимость	недорого	дорого

Таким образом, если суммировать всю данную главу, то везде, где применяется распознавание речи или изображений, робототехника, устный или письменный перевод, чат-боты, беспилотное вождение транспортных средств, предсказание каких-то параметров на основе имеющихся данных, во всех этих примерах присутствуют элементы ИИ, потому что ИИ – это

очень широкое понятие, которое охватывает все эти направления, когда компьютер имитирует мышление и поведение человека.

Случаи, когда мы вместо того, чтобы давать компьютеру написанные инструкции и правила для решения вопроса, даем ему набор данных и он сам учится на них и находит необходимые алгоритмы и закономерности самостоятельно, такие случаи называются Машинным обучением. И одним из вариантов нахождения компьютером таких закономерностей является глубокое обучение, в котором используется несколько слоев нейронных сетей, что делает такие вычисления с одной стороны, более эффективными, с другой стороны, более трудными для расшифровки.

Примеры использования ИИ, МО и ГО

Давайте посмотрим несколько примеров использования Искусственного интеллекта, машинного и глубокого обучения в нашей повседневной жизни.

Искусственный интеллект

Все лунные модули, которые бороздят поверхность Луны, используют алгоритмы ИИ. Их не надо контролировать каждую секунду, они сами принимают решения как объезжать препятствия и как собрать грунт в том или ином труднодоступном месте.

ИИ применяется и в беспилотных автомобилях. С помощью множества сенсоров, такие автомобили анализируют находящуюся вокруг них обстановку, определяют другие движущиеся машины, пешеходов, знаки дорожного движения, разметку, выбирают кратчайший путь и т.д.

Наше взаимодействие с голосовыми помощниками. Когда мы просим Алексу, Сири, или Алису от Яндекса сделать или найти что-то, они конвертируют наш голос в команды, обрабатывают их и выдают то, что нам необходимо.

Кроме голосовых помощников, очень развиты сейчас чат-боты, когда вы можете переписываться с компьютером, и он будет отвечать на ваши запросы. А в последнее время участились и звонки роботов на наши мобильные телефоны. Они могут предлагать какие-то рекламные акции или даже расспрашивать у вас информацию, например, когда вы планируете погасить кредитную задолженность. Такие роботы уже заменили многих сотрудников колл-центров.

Машинное обучение

Улучшение выдачи результатов поиска в Google. Когда ты вбиваешь какой-то запрос в поисковой строке, тебе выводится несколько ссылок. Если ты заходишь по одной из ссылок на первой странице, и просматриваешь страницу и проводишь там какое-то время изучая и читая информацию на этой странице, Google понимает, что ты нашел что искал. Когда заходишь на вторую, третью страницу, и видишь, что все это не то, то Google понимает, что это менее нужная информация, и в следующий раз когда другой человек зайдет на Google и спросит его об этом же, то Google будет знать, что лучше выдать в первой строчке на первой странице.

Решение о выдаче кредита банком. Компьютер анализирует большое количество параметров потенциального заемщика и потом распределяет его в категорию хороший или плохой заемщик, либо дает ему конкретный кредитный скоринг. Все это происходит на основе кредитной истории предыдущих заемщиков и как они схожи с потенциальным новым заемщиком. Выборка постоянно дополняется историей каждого нового заемщика, расплатился ли он с кредитом и выплатил ли его вовремя, она обновляется, и также обновляется и алгоритм, находятся новые закономерности, которые позволяют принимать правильные решения о выдаче кредита новому заемщику.

Выбор места для ритейла. В ритейле одним из самых главных факторов, которые влияют на прибыльность бизнеса, является местоположение. У сети кофеен Старбакс имеется около 30 000 локаций по всему миру. Вы накопили большой объем информации о том, в каких местах продажи лучше. На основе этой информации вы можете составить алгоритм по выбору наиболее удачного места в новой локации. Ваши переменные могут включать геохарактеристики (расстояние до центра города, до метро, цена за квадратный метр), трафик (число маршрутов наземного транспорта в разных радиусах от локации) и наличие тех или иных объектов рядом, например, торговых центров, бизнес-центров, домов, школ и банков.

Глубокое обучение

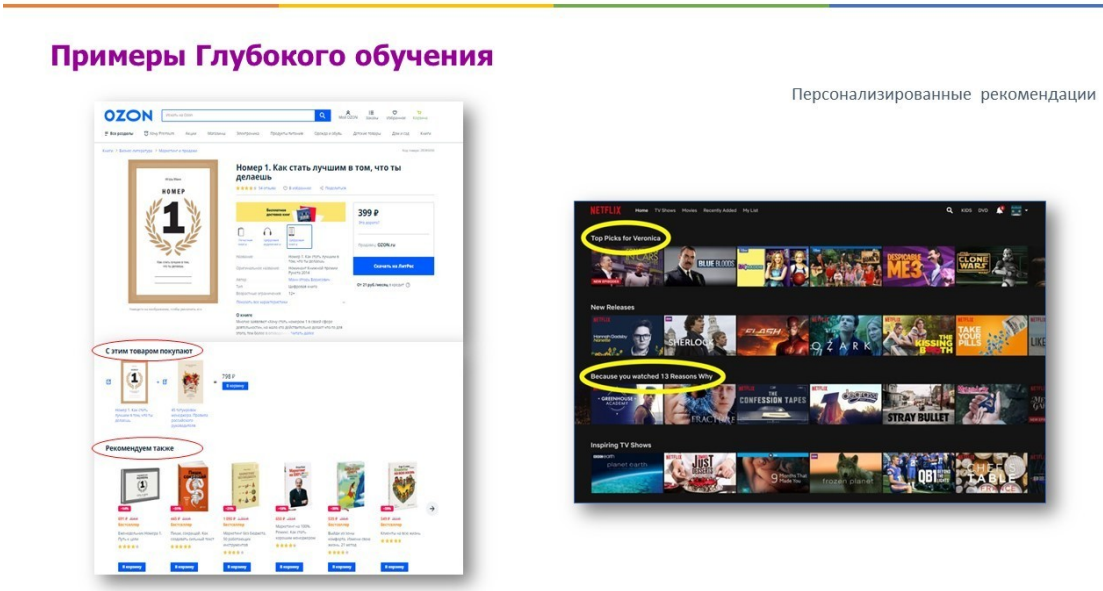
Очень часто ГО используется для распознавания объектов на изображениях. Кроме того, с помощью ГО черно-белые изображения или фильмы можно сделать цветными. До этого компьютер уже обработал большое количество данных и информации в интернете либо в базе данной, которую ему предоставили для этого, и он уже знает различные оттенки серого и может легко понять в какой цвет необходимо преобразить тот или иной пиксель изображения.

Машинный перевод. Возможно, кто-то из вас использовал Google Translate, и вы могли заметить насколько хорошо он переводит в последнее время. Практически ничего не надо исправлять. Но если вспомнить примерно 5 или 7 лет назад, то качество перевода было далеко от идеального. А все потому, что сейчас вместо множества правил как надо переводить, используются нейронные сети, через которые уже прошли миллионы переводов художественной, технической и другой литературы, и эти алгоритмы ГО все продолжают улучшаться.

Интеллектуальные игры: шахматы, Го, Дота 2, покер и другие. Долгое время считалось, что компьютер никогда не превзойдет по силе мысли человека до тех пор, пока он не сможет обыграть его в шахматы. Однако, это случилось в конце 20 века, а в 2010-х годах, компьютер, обученный алгоритмами ГО, смог обыграть и чемпионов в го – игру, которая считается даже еще более сложной чем шахматы. Сейчас не проходит и года, как не появляется очередная новость о том, что компьютер обыграл человека в очередной игре. ИИ уже обыграл людей в покер, Доту 2 и другие интеллектуальные игры. Все это получилось благодаря задействованию нейронных сетей и ГО.

Распознавание злокачественных заболеваний на коже или органах человека. Одним из самых полезных применений ИИ – это медицина. С помощью ГО и нейронных сетей компьютеры сегодня могут распознавать злокачественные опухоли еще на ранней стадии и даже лучше, чем опытные доктора. Это хорошо еще и тем, что пациент, находящийся в одной точке земного шара, может переслать свои снимки в лабораторию в другой стране для принятия решения. Предсказывается, что в будущем роботы с помощью ИИ будут выполнять все больше и больше сложных операций без участия человека.

Еще одним популярным применением ГО являются так называемые рекомендательные системы: когда при покупке одного товара нам предлагают другой. Наверное, вы видели, когда на сайте появляется фраза: «с этим товаром часто покупают». Или при просмотре фильма, или книги на сайте агрегаторе, вам начинают предлагать фильмы и книги похожей категории или те фильмы, которые смотрели пользователи, похожие на вас по различным параметрам. Все это алгоритмы ИИ, подкрепленные НС.



И в конце, на что еще хотелось бы обратить внимание. Как уже было сказано, и ГО и МО являются только частью более общей области под названием ИИ. Так вот, в сложных проектах, как правило, присутствует несколько видов алгоритмов ИИ, и глубокое обучение и машинное обучение, и другие виды. Например, во время движения беспилотного автомобиля участвует более 100 различных алгоритмов, которые ответственны за распознавание объектов, управление движением, навигацию, безопасность, и т.д.

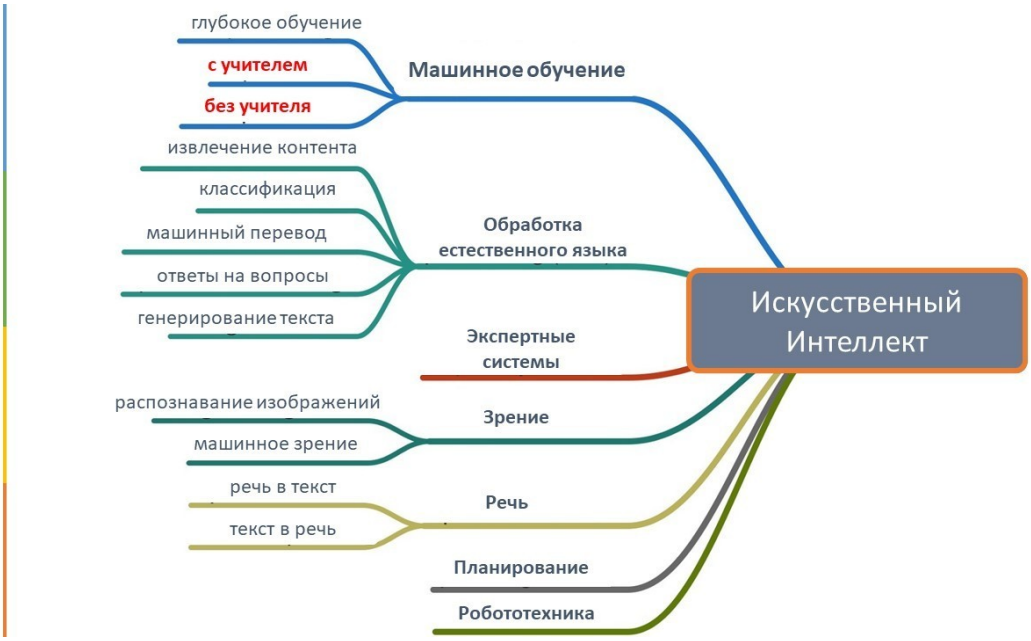
Как вы заметили по приведенным примерам, ИИ уже используется во многих областях в нашей повседневной жизни. Считается, что в ближайшие пару десятилетий ИИ будет использоваться большинством компаний и охватывать большую часть нашей жизнедеятельности.

Основные задачи и методы машинного обучения

Обучение с учителем и обучение без учителя

Если вы интересовались темой искусственного интеллекта и машинного обучения, возможно вы уже встречались с такими понятиями как обучение с учителем (на англ. supervised learning) и обучение без учителя (unsupervised learning). В этой главе мы узнаем, чем отличаются эти два понятия.

Во-первых, они оба являются видами машинного обучения.

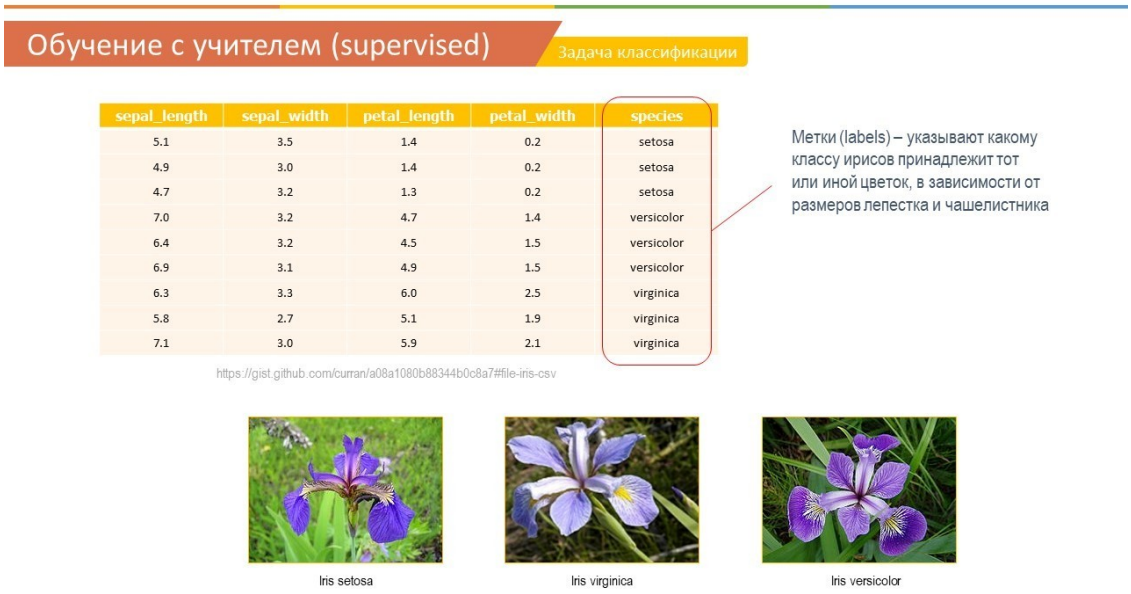


Во-вторых, обучение с учителем не обязательно подразумевает, что кто-то стоит над компьютером и контролирует каждое его действие. В терминах машинного обучения, обучение с «учителем» означает, что человек уже подготовил данные для дальнейшей работы над ними компьютером, то есть у каждого объекта имеется метка (на англ. label) которая выделяет этот объект от остальных объектов или дает ему какое-то именное или числовое наименование. И компьютеру остается только найти закономерности между признаками объектов и их наименованиями, основываясь на этих подготовленных или как их называют помеченных данных. На английском такие данные называются labeled data.

Обучение с учителем включает два основных типа задач: регрессия и классификация. Давайте посмотрим на типичный пример задачи классификации.

Это будет пример цветков ириса Фишера. Этот набор данных стал уже классическим, и часто используется для иллюстрации работы различных статистических алгоритмов. Вы можете найти его по следующей ссылке (https://gist.github.com/curran/a08a1080b88344b0c8a7) либо просто вбив в интернете.

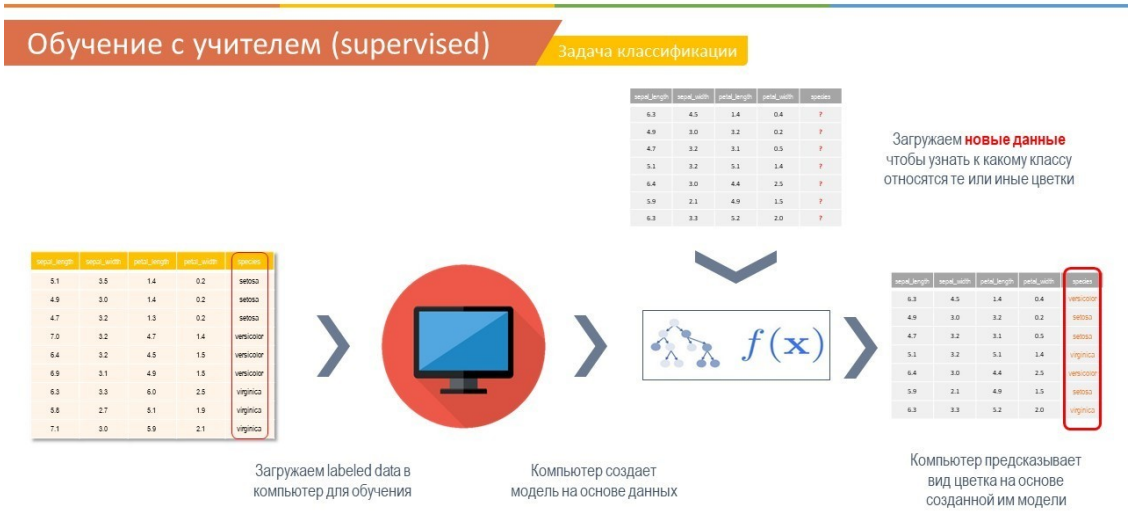
В природе существует три вида цветков ириса. Они отличаются друг от друга размерами лепестка и чашелистника. Все данные по цветкам занесены в таблицу, в столбиках указаны длина и ширина лепестка, а также длина и ширина чашелистника. В последнем столбце указан вид ириса – Ирис щетинистый (*Iris setosa*), Ирис виргинский (*Iris virginica*) и Ирис разноцветный (*Iris versicolor*). Тот или иной вид ириса и является в нашем случае меткой.



sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica
7.1	3.0	5.9	2.1	virginica

На основании этого набора данных требуется построить правило классификации, определяющее вид растения в зависимости от размеров. Это задача многоклассовой классификации, так как имеется три класса – три вида ириса.

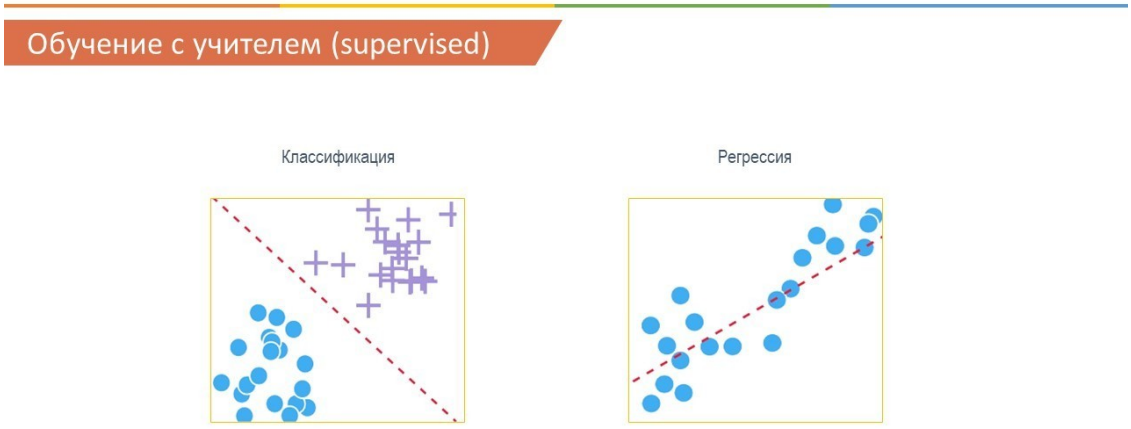
В данном случае с помощью алгоритма классификации, мы разделяем наши ирисы на три вида в зависимости от длины и ширины лепестка и чашелистника. В следующий раз, если нам попадется новый представитель ирисов, с помощью нашей модели мы сможем сразу же его поместить в тот или иной из трех классов.



Почему этот пример можно считать обучение с учителем? Потому что наши данные распределены по признакам, у каждого признака есть показатель для конкретного цветка, то есть размеры длины и ширины. И имеются ответы или метки, какой вид ириса бывает при тех или иных размерах лепестка и чашелистника. То есть мы как учитель обучаем нашу модель и говорим ей, что вот окей, если ты видишь, что размер лепестка такой-то, а чашелистника – такой, то этой ирис виргинский, а если размеры такие-то и такие-то, то это ирис разноцветный. Это и называется обучение с учителем, когда мы показываем нашей модели все ответы в зависимости от признаков. Модель учится на этих данных, и создает формулу или алгоритм, который поможет нам в будущем предсказывать вид цветка в зависимости от размеров, когда нам будут поступать новые образцы цветов.

Кроме задач классификации, о которой мы только что говорили в примере с ирисами, есть еще один вид машинного обучения с учителем. Это регрессия.

Если в задачах классификации мы имеем несколько классов объектов, то в задачах регрессии, у нас один класс, но каждый объект отличается от другого и нам надо предсказать какой будет числовой показатель того или иного признака каждого объекта в зависимости от других его признаков и опять же на основании набора данных, которые мы предоставим нашему компьютеру.



Классический пример регрессии – это когда мы предсказываем цену квартиры в зависимости от ее площади.

Опять же мы имеем какую-то таблицу с данными разных квартир. В одном столбце площадь, а в другом – цены на эти квартиры. Это очень упрощенный пример регрессии, естественно, что цена квартиры будет зависеть от множества других факторов, но все же он наглядно демонстрирует, что такое регрессия. Так вот, в последнем столбце мы расположили фактические или реальные цены на квартиры с таким метражом. То есть, мы как учитель, показываем нашей модели, что вот, если видишь, что метраж такой-то, то цена будет такая-то и т.д. На основе этих данных модель учится, и потом выдает алгоритм, на основе которого мы можем предсказывать, какая будет цена квартиры, если условная площадь будет такая-то.

Таким образом, если суммировать, то в обучении с учителем – ключевая фраза – это labeled data или помеченные данные. То есть мы загружаем в нашу модель данные с ответами,

будь то класс, к которому принадлежит тот или иной объект или реальная цена квартиры в зависимости от площади. На основе этой информации модель учится и создает алгоритм, который может делать прогнозы.

Идем дальше. Второй вид машинного обучения – это **обучение без учителя**. Это когда мы позволяем нашей модели обучаться самостоятельно и находить информацию, которая может быть не видна очевидно для человека.

В отличие от обучения с учителем, модели, которые используются в обучении без учителя, выводят закономерности и выводы на основе немаркированных данных (или unlabeled data). Помните, у нас был пример с цветками ириса. Так вот в данных, которые мы давали компьютеру, присутствовали ответы какой вид ириса мы имеем в зависимости от тех или иных размеров лепестка и чашелистника. А в немаркированных данных, у нас имеются данные и признаки, но мы не имеем ответа к какому виду или классу они относятся. Поэтому такие данные называются немаркированные.

В обучении без учителя основными типами задач являются Кластеризация и снижение размерности. Если в двух словах, то снижение размерности означает, что мы удаляем ненужные или излишние признаки из наших данных, чтобы облегчить классификацию наших данных и сделать ее более понятной для интерпретации.

Давайте рассмотрим пример **кластеризации**.

В задачах кластеризации у нас имеется набор объектов и нам надо выявить его внутреннюю структуру. То есть нам надо найти группы объектов внутри этого набора, которые наиболее похожи между собой, и отличаются от других групп объектов из этого же набора. Например, разобрать все движущиеся средства по категориям, например, все средства, похожие на велосипед, в одну группу или кластер, а похожие на автобус – в отдельную группу. Причем, мы не говорим компьютеру, что чем является, он должен самостоятельно найти схожие признаки и определить похожие объекты в ту или иную группу. Поэтому это и называется обучение без учителя, потому что мы не говорим изначально компьютеру к какой группе принадлежат те или иные объекты.

Такие задачи бывают очень полезны для крупных ритейлеров, если они, например, хотят понять из кого состоят их клиенты. Предположим, есть крупный гипермаркет, и чтобы делать точечные рекламные акции для своих потребителей, ему необходимо будет разбить их по груп-

пам или кластерам. И если сейчас акция на спортивные товары, то отправлять информацию об этой акции не всем подряд потребителям, а только тем, кто в прошлом уже покупали спортивные товары.

Таким образом, основная разница между обучением с учителем и обучением без учителя, это то, что в обучении с учителем мы используем маркированные данные, где каждый объект помечен и относится к тому или иному классу или имеет конкретное числовое значение. И на основе этих помеченных данных наша модель строит алгоритм, который помогает нам прогнозировать результаты при новых данных. А в обучении без учителя, имеющиеся у нас данные непромаркированы, и компьютер самостоятельно выводит определенные закономерности и общие признаки и разделяет все объекты на разные группы, схожие внутри одной группы и отличающиеся от объектов в других группах.

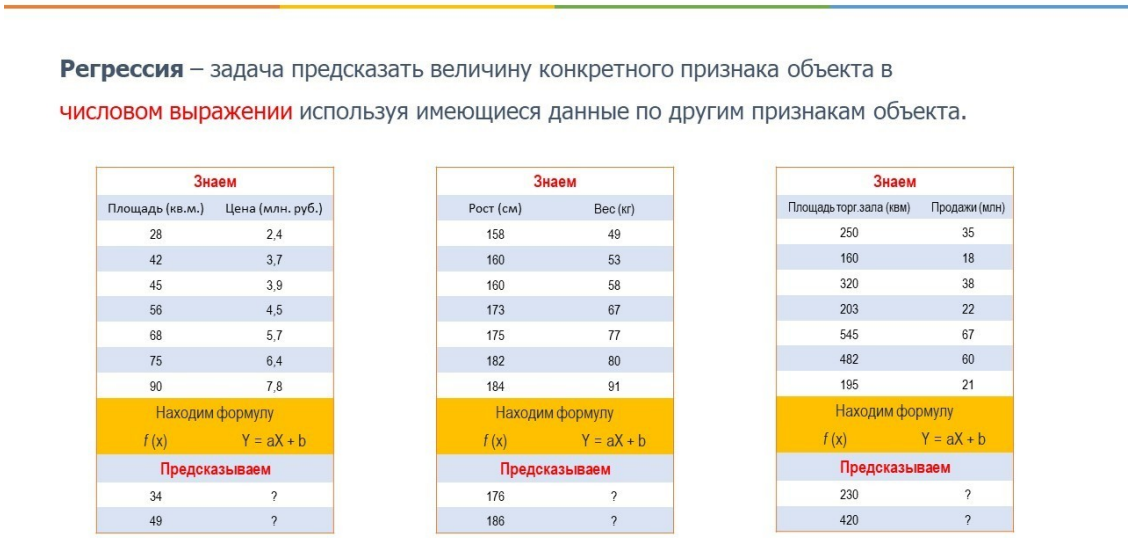
Основные задачи **обучения с учителем** разделяются на два типа: Классификация, когда мы разделяем наши данные на классы, и Регрессия, когда мы делаем численный прогноз на основе предыдущих данных.

Основные задачи **обучения без учителя** включают в себя кластеризацию, когда компьютер делит наши данные на группы или кластеры. И снижение размерности, которое необходимо для более удобной демонстрации больших объемов данных.

Указанные задачи мы рассмотрим более подробно в следующих главах.

Регрессия

Итак, одной из самых популярных задач машинного обучения является регрессия. Это задача определить какую-то величину в цифрах (например, вес человека, стоимость квартиры, объем продаж) используя известную информацию (рост, площадь, удаленность от метро, сезонность).

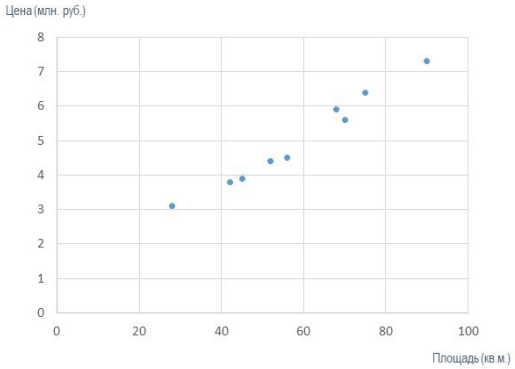


Давайте возьмем пример с предсказанием стоимости квартиры в зависимости от площади. Для любой задачи машинного обучения нужны данные, и чем больше, тем лучше. Так вот, представим, что у нас есть табличка с данными, в одном столбце площадь квартиры, в другом цена этой квартиры.

Мы располагаем эти данные на графике и в принципе можем заметить, что тут имеется определенная линейная зависимость, которая достаточно очевидна, что чем больше площадь, тем выше стоимость квартиры. Понятное дело, что на стоимость квартиры будет влиять намного больше факторов, как например, удаленность от центра города и от метро, этажность, возраст дома и т.д. Но для упрощения, возьмем всего один признак – площадь квартиры.

Задача регрессии:
Определить стоимость квартиры в зависимости от площади

Знаем	
Площадь (кв.м.)	Цена (млн. руб.)
28	3,1
42	3,8
45	3,9
52	4,4
56	4,5
68	5,9
70	5,6
75	6,4
90	7,3
Находим формулу зависимости $f(x)$	
Предсказываем	
34	?
49	?



Так вот, наша задача – научиться предсказывать цену. Для этого нам нужна будет формула, с помощью которой мы сможем подставлять площадь, и нам будет выдаваться цена.

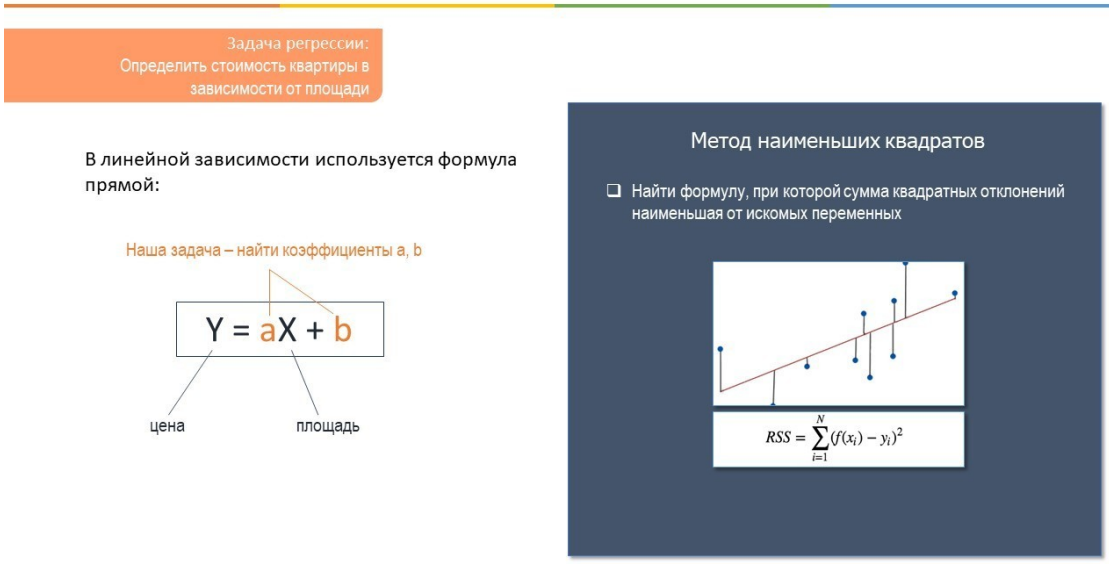
В данном случае мы видим линейную зависимость, и в таких ситуациях используется **формула прямой Y = AX + B**, в которой Y = цена, X – площадь.

На самом деле, зависимость необязательно будет линейной, она может быть кривой, либо иметь совсем странный вид.

Так вот, чтобы у нас была конкретная рабочая формула, нам надо найти коэффициенты A и B.

Как это можно сделать? Самый простой классический способ, который вы наверняка проходили на уроках алгебры или статистики – это **метод наименьших квадратов**. На самом деле этот метод был придуман еще 200 лет назад, и сейчас появились более эффективные решения, но тем не менее метод наименьших квадратов по-прежнему актуален и используется достаточно часто в задачах регрессии.

Метод наименьших квадратов заключается в том, что вы находите такую формулу, при которой сумма квадратных отклонений наименьшая от искомых переменных.

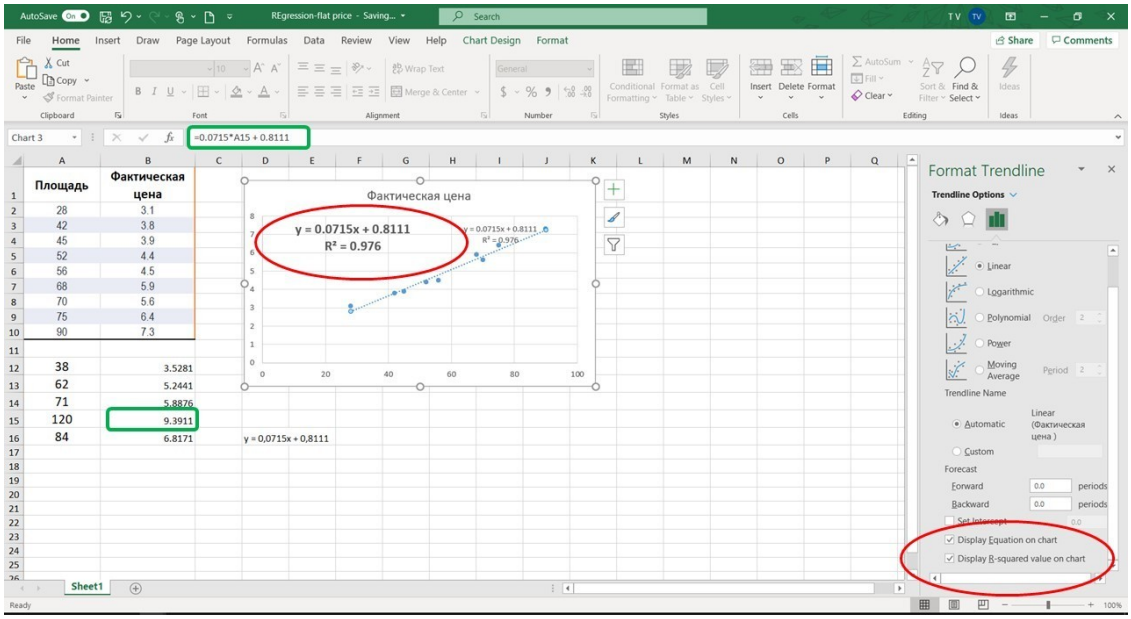


Давайте посмотрим, как это может выглядеть в Excel. Откроем файл, в котором у нас имеется таблица с некоторым количеством квартир, их площадью и ценой. Наложим эти данные на график и получим точки с соответствующими значениями. Мы видим, что тут имеется определенная зависимость, что чем больше площадь, тем выше стоимость квартиры.

Давайте проведем линию, в Excel это можно сделать следующим образом. Кликаем на график, сбоку появляется плюсик, кликаем на него и ставим галочку показать трендлайн. Ок, линия видна. Но как же найти конкретные коэффициенты нашей формулы, чтобы мы могли предсказывать цены для новых квартир.

По идее, можно сделать долгие расчеты, и решить это уравнение здесь же в Excel с помощью матричных вычислений. Однако, это можно сделать намного проще в два клика. Опять же нажимаем на плюсик, кликаем на дополнительные опции. И здесь ставим галку показать формулу.

И вот мы уже видим нашу формулу в верхней части графика. Y = 0,0715x + 0,811. Внизу показана буква r, она обозначает **R-squared values**.



Этот показатель принимает значения от 0 до 1 и подразумевает точность от 0 до 100%. Если значение 1, то это значит, что наша линия или наша формула на 100% правильно описывает соотношение наших показателей, в нашем случае соотношение цены и площади. Как видим на этом графике, показатель R является 0,976, что достаточно высокий показатель, который говорит о том, что наша формула очень эффективна.

Соответственно, теперь мы можем предсказывать цены в зависимости от площади. Зная эту формулу и коэффициенты, мы просто подставляем площадь какой-то новой квартиры и видим, какая должна быть цена в соответствии с нашей формулой.

Как вы понимаете, эта формула также будет меняться. Каждый раз, когда вы вносите какие-то новые данные реальных квартир с реальными ценами, она будет изменяться и подстраиваться таким образом, чтобы максимально соответствовать всем объектам в нашей выборке.

Еще один момент. Вы можете настроить вашу формулу регрессии таким образом, чтобы старые цены влияли на вашу формулу меньше, потому что очевидно, что они являются менее актуальными. А новые цены квартир имели бы больший вес в вашей выборке.

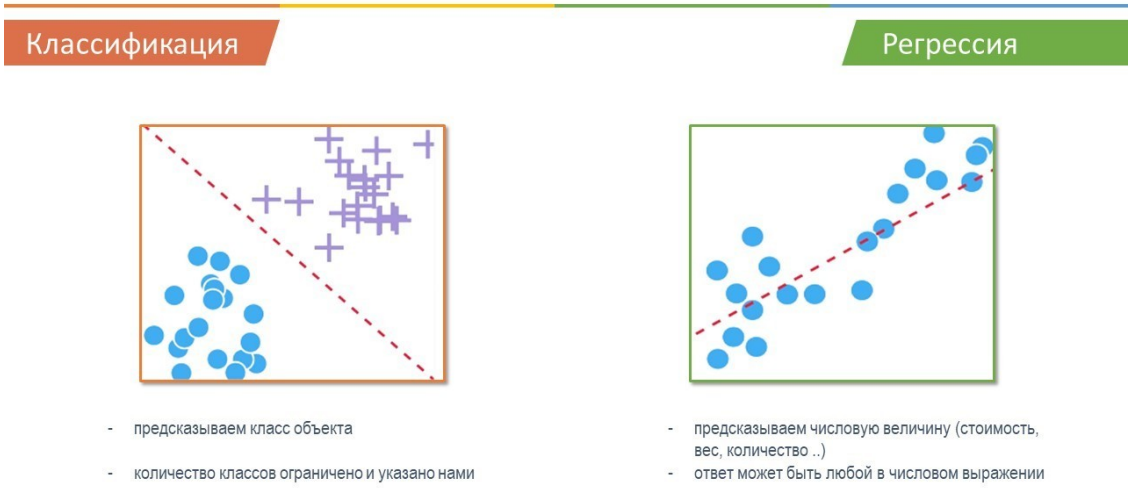
Таким образом, если суммировать. В задачах регрессии мы предсказываем конкретное числовое значение какого-то признака, используя данные других признаков. И все это мы делаем с помощью формулы, которую мы выводим на основе уже имеющегося набора реальных данных.

Классификация

Классификация (тип задач обучения с учителем) представляет собой большой класс задач, которые используются так же часто, как и регрессия.

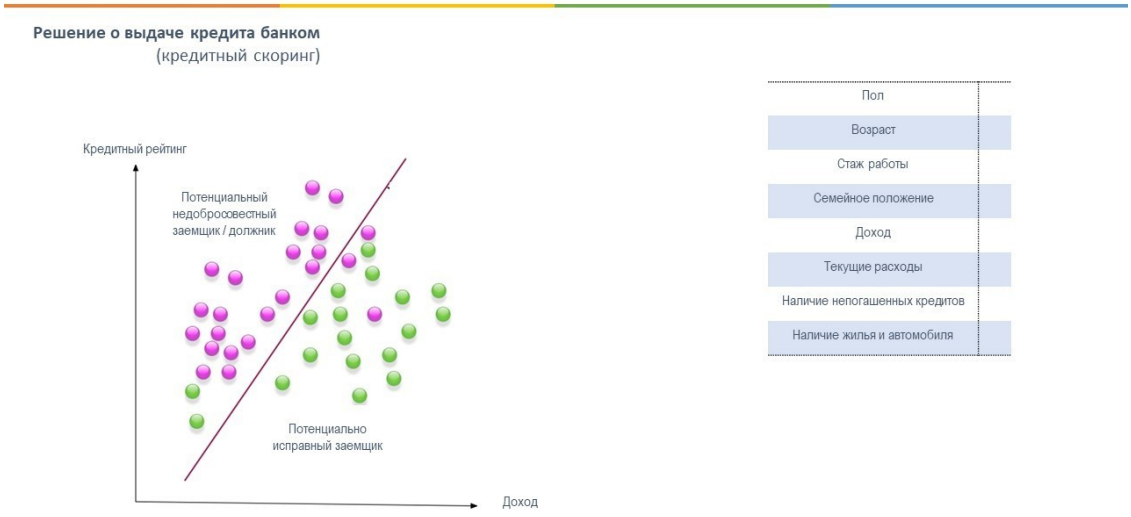
Как можно понять из названия, классификация используется для того, чтобы отнести тот или иной объект к определенному классу.

Если сравнить с регрессией, то в регрессии у нас нет классов, мы просто предсказываем числовую величину. А в классификации количество классов ограничено, мы сами предоставляем эти классы компьютеру, и компьютер определяет к какому из этих классов относится новый объект.



Давайте посмотрим несколько реальных кейсов, когда могут использоваться задачи классификации.

Многие банки сегодня, чтобы определить выдавать кредит тому или иному заемщику, используют алгоритмы классификации. В такой задаче имеется как правило два класса: первый класс – заемщик платежеспособный, и второй класс – потенциально недобросовестный заемщик. Теперь представим в банк приходит новый клиент и хочет взять кредит. Как банки определяют к какому классу относится тот или иной потенциальный заемщик, особенно если у них не было предыдущего опыта взаимодействия с данным конкретным человеком? Правильно, они смотрят на большую базу данных своих клиентов и делают выводы на какого типичного их клиента похож этот новый заемщик.

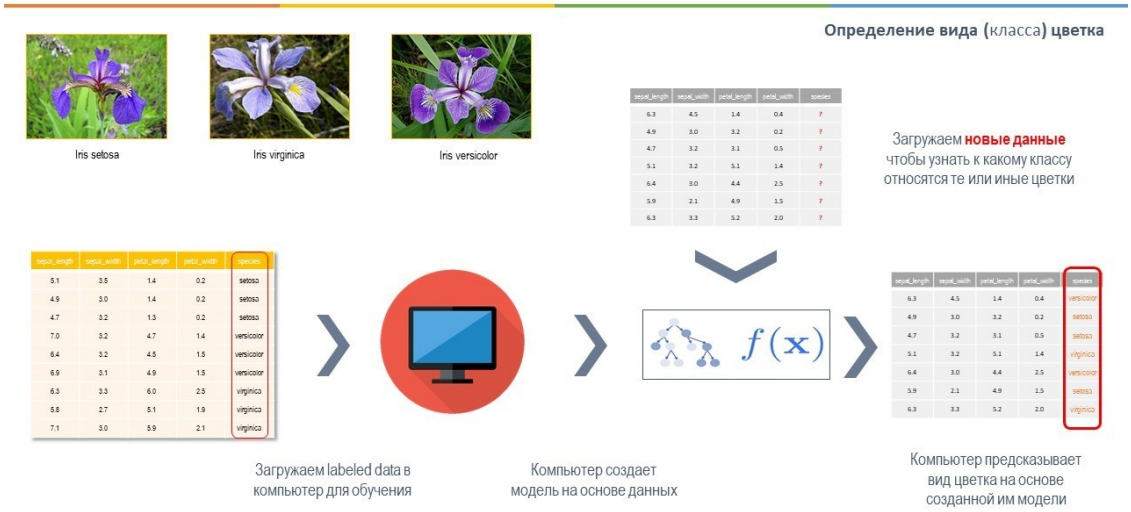


Как работают эти алгоритмы классификации. У банка имеются десятки или даже сотни тысяч данных их клиентов, и они знают какие из клиентов платят вовремя, а какие задерживают оплату или даже становятся неплатежеспособными. Они агрегируют эти данные по признакам, например, тот у кого сумма кредита не превышает 30% от его ежемесячного дохода, как правило, платит исправно, а тот, у кого уже есть другой кредит, часто задерживает. И таких логических цепочек очень много. На основе этой информации компьютер в банке строит модель классификации. Имеется несколько разных моделей, которые можно использовать для задач классификации. Мы их рассмотрим более подробно в следующих лекциях.

Так вот компьютер строит модель, потом банковский менеджер просит заполнить анкету у нового клиента, далее он вводит все эти данные в модель, и алгоритм выдает ответ, этот новый клиент является потенциально добросовестным или недобросовестным заемщиком. Модель даже может указать с какой вероятностью он будет задерживать кредит и будет ли вообще задерживать его.

Сейчас так называемый кредитный скоринг в большинстве банков осуществляется автоматически даже без привлечения банковских специалистов. Клиенты просто заполняют заявку онлайн, и компьютер автоматически дает одобрение или нет. В сложных и спорных случаях привлекается банковский менеджер.

В предыдущей лекции мы уже приводили пример цветков Ириса, это тоже типичная задача классификации. Мы имеем три класса цветков, и нам надо научиться предсказывать к какому классу относится тот или иной цветок в зависимости от размеров его лепестков и чашелистника.



Задачи классификации можно решить с помощью разных методов. Наиболее часто используемыми являются следующие:

– Дерево решений

– Логистическая регрессия (не путайте с обычной регрессией, о которой мы уже упоминали, и где мы предсказываем конкретное числовое значение.) В логистической регрессии немного по-другому, с помощью алгоритмов мы находим такую линию, которая разделяет наш набор данных на классы.

– Случайный лес

– Ансамбли и бэггинг

– Метод опорных векторов

– Метод К-ближайших соседей

Мы рассмотрим эти методы более подробно в следующих лекциях. Пока же, если суммировать, то в задачах классификации мы предсказываем к какому классу относится тот или иной объект, количество классов ограничено (то есть например, выдать кредит или нет, на картинке изображен автобус, автомобиль или велосипед, и т.д.).

Сейчас Искусственный интеллект в задачах классификации развивается очень быстро, пожалуй, даже быстрее чем регрессия, и поэтому изучение методов классификации также представляет собой достаточно актуальную задачу.

Метод

K

–ближайших соседей. Решение задачи классификации в

Excel

Итак, давайте посмотрим, как выглядит решение задачи классификации с помощью метода К-ближайших соседей. Мы будем решать данную задачу в Excel, и вы можете повторить ее вместе со мной, я буду прикреплять скрины с данными в этой главе.

Итак, перед нами уже известная нам таблица с цветками Ириса Фишера, которую мы упоминали в предыдущих лекциях. Для упрощения мы взяли всего 15 разных цветков и только два признака – длина и ширина чашелистника, в третьей колонке – наша метка или label – то есть ответы, к какому виду цветков принадлежит тот или иной отдельно взятый цветок. Всего три вида цветков. И эта задача классификации, потому что нам надо определить, к какому

виду или классу из трех возможных принадлежит наш новый цветок, который как раз указан внизу таблицы. Мы видим его размеры, но не знаем его вида.

	A	B	C
1	sepal_length	sepal_width	species
2	5.1	3.8	setosa
3	4.8	3	setosa
4	5	3.5	setosa
5	4.7	3.2	setosa
6	5.3	3.7	setosa
7	6.2	2.2	versicolor
8	5.6	2.5	versicolor
9	5.9	3.2	versicolor
10	6.1	2.8	versicolor
11	6.3	2.5	versicolor
12	7.7	2.6	virginica
13	6.9	3.2	virginica
14	6.3	2.7	virginica
15	7.2	3.2	virginica
16	7.3	2.9	virginica
17			
18	5.7	2.9	???
19			

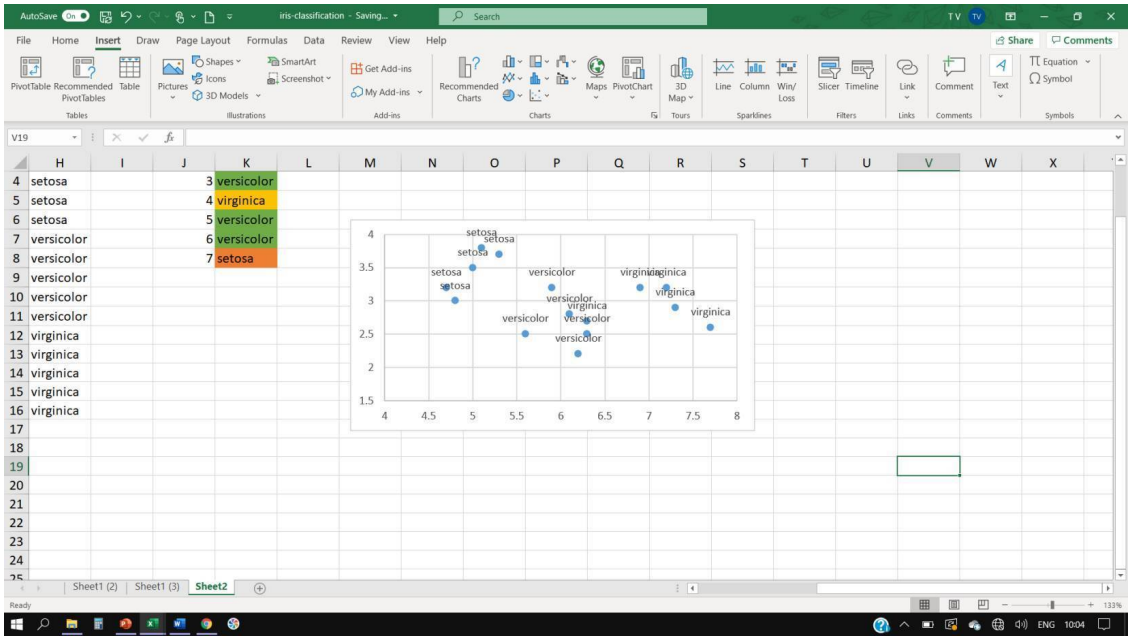
Итак, для начала, давайте построим график, чтобы выглядело более наглядно.

Выбираем колонки A и B, заходим во вкладку “Вставить”, и выбираем график разброса (Insert Chart – Scatter). Мы видим, как наши точки расположились на этой сетке в зависимости от длины и ширины чашелистника.

Мы можем убрать лишние пустые места в этом графике. Начнем с оси Y. Кликаем на цифры по оси игрек. Сбоку выходят свойства, и мы ставим минимум, ну, возьмем, наверное, полтора. Так, график немного подвинулся, снизу ушли пустые ненужные клетки. Делаем то же самое с осью X. Здесь уберем пустое поле от нуля до четырех.

Отлично, теперь график выглядит получше. Видите, уже сейчас заметно, что есть три отчетливые группы точек. Это и есть наши три вида цветков Ириса.

Давайте добавим названия видов над каждой точкой. Для этого кликаем на график, потом на появившийся плюсик, и здесь ставим галочку Data labels. У нас появились числа ширины над каждой точкой. А нам надо их названия. Поэтому кликаем на дополнительные опции и здесь выбираем Value from Cells, то есть придаем им значения из конкретных ячеек, и в появившемся окошке выбираем наш третий столбик с видами цветков.



Отлично, у нас появились названия над каждой точкой, можем снять галочку с отображением ширины. Да, у нас отчетливое разделение на три группы, это можно заметить, оно достаточно правильное, единственное, один цветок virginica каким-то образом оказался среди цветков versicolor, но такое бывает, это называется статистический выброс.

Теперь давайте разместим наш новый цветок, вид которого мы не знаем на нашем графике. Для этого кликаем на график, заходим во вкладку “Дизайн”, здесь нажимаем вкладку “Выбрать данные” и в появившемся окне, нажимаем добавить и вставляем ячейки с размерами нового объекта.

Нажимаем ОК, и вот наш новый цветок появился на нашем графике рядом с другими цветками. Давайте тоже уберем сверху его ширину. Отлично.

Итак, метод К-ближайших соседей заключается в том, чтобы определить какие объекты являются ближайшими соседями нового объекта. И соответственно, новый объект будет принадлежать к тому классу, ближайших соседей из которого будет больше к данному объекту.

Данный пример достаточно простой, здесь всего 15 объектов и всего два признака – ширина и длина. И в принципе, по графику понятно, что ближайшими соседями являются цветки вида versicolor. Но представьте, что было бы если бы у нас было тысячи или даже десятки или сотни тысяч объектов и у каждого было бы по несколько различных признаков. В таком случае простой график нам бы не помог, и мы бы даже возможно не смогли разместить все данные на одном графике, если бы было много признаков, а значит несколько плоскостей и координат.

Тем не менее данный пример достаточно показательный, и поможет нам понять, как пользоваться методом К-ближайших соседей.

Сделаем колонку под названием Евклидово расстояние. Это расстояние в многомерном пространстве. В нашем примере есть два признака или две оси на графике, длина и ширина чашелистника, и нам потребуется найти расстояние от нашего нового цветка до всех остальных цветков по двум осям.

Для определения Евклидова расстояния используется следующая формула: Корень из суммы квадратов разницы расстояний по всем осям, в нашем случае две оси.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Пишем в первой ячейке равно, потом sqrt, дальше открываем скобки и еще одни, потому что у нас будет две разницы. Внутри скобки вставляем ячейку A2, где длина первого цветка, дальше минус ячейку A18, где у нас длина искомого цветка. И очень важно, после этого нажмите кнопку F4 на компьютере, чтобы зафиксировать ячейку A18, потому что мы будем вычитать ее из всех цветков. После нажатия кнопки F4 у вас должны появиться символы доллара перед буквой A и 18. Дальше возводим эту первую скобку в квадрат и прибавляем квадратную разницу между шириной первого цветка и искомого цветка. Не забудьте также зафиксировать ширину искомого цветка, т.е. ячейку B18 с помощью кнопки F4.

Вот какая формула в итоге у нас должна получиться:

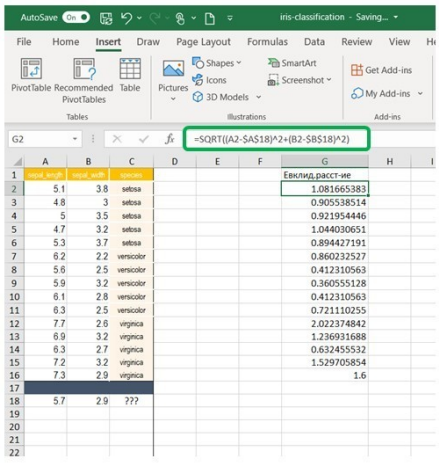
=SQRT((A2-A18)^2+(B2-B18)^2)

Итак, евклидово расстояние у нас получилось 1,08, что в принципе выглядит правдивым, если посмотреть на график. Теперь давайте продублируем эту формулу на все 15 цветков.

Это делается следующим образом. Кликните на ячейку с расстоянием, наведите курсор на крайний правый нижний угол ячейки, увидите крестик, нажмите и держите его и опустите этот крестик вниз до последней ячейки.

Отлично, все готово. Теперь у нас есть расстояния между нашей искомой точкой и всеми остальными цветками.

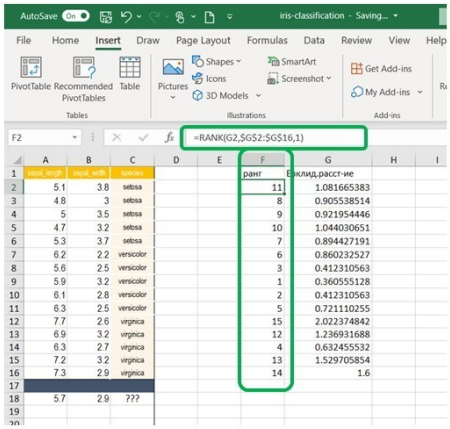
Как мы видим, самое короткое расстояние, это у 9-й ячейки, всего 0,36, а самая далекая точка – это 12 ячейка – здесь расстояние больше 2. Скорее всего, это цветок вирджиника, который расположен дальше всех.



Но мы смогли найти самую близкую точку, потому что у нас всего 15 объектов в выборке. А что, если таких объектов тысячи. Для этого нам надо чтобы компьютер автоматически проранжировал все эти расстояния и указал, которая ячейка самая близкая, и так по порядку.

Для этого в соседней колонке, которую мы назовем ранг, давайте напишем следующую формулу. Равно, потом слово RANK, скобки открываем, кликаем на первую ячейку G2 с расстоянием. Дальше точка с запятой, и теперь выбираем диапазон всех ячеек с расстояниями, то есть мы говорим, скажи какой ранг занимает ячейка G2 среди всех этих ячеек. Не забудьте нажать после этого клавишу F4, чтобы зафиксировать этот диапазон ячеек, у вас должны появиться значки доллара в формуле. Дальше после точки с запятой, нажимаем 1 и закрываем скобки. Единицу мы ставим, чтобы нам ранжировали от меньшего расстояния к большему. Нажимаем Enter и вот у нас появился ранг 11. То есть этот цветок находится на 11 месте по близости к нашему искомому цветку.

Итак, а теперь давайте посмотрим на ранги остальных точек. Для этого нажимаем на крестик в первой ячейке и тянем его вниз до последней ячейки.



Отлично, теперь у нас есть все расстояния по мере возрастания. Давайте укажем в соседнем столбце, какие виды цветков мы имеем.

И теперь последний шаг.

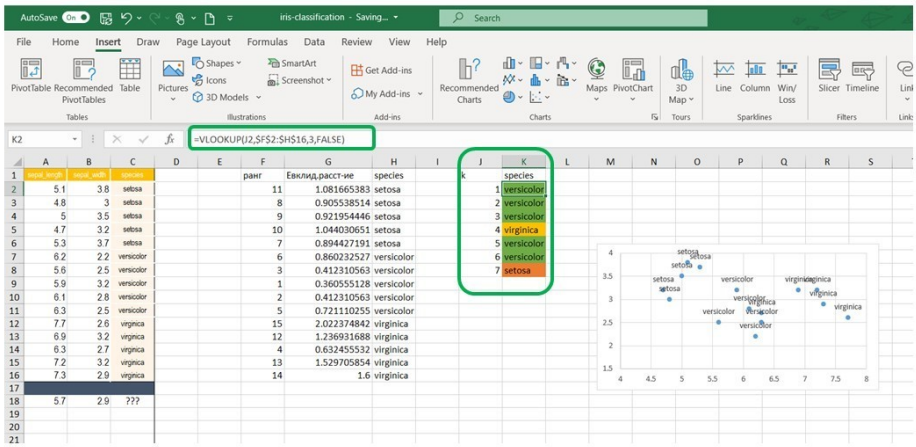
Данный метод называется – метод К-ближайших соседей. Под буквой «К» подразумевается какое-то число, на основе которого мы будем принимать решение. Возьмем, например, число три. Это будет означать, что мы будем смотреть на трех ближайших соседей и в зависимости от того, цветков какого вида ближе, мы и примем решение, к какому виду относится наш искомый цветок.

Теперь нам надо подставить тот вид цветка, который соответствует рангу 1, 2 или 3. В данном случае мы, конечно, можем сделать это от руки, потому что у нас всего 15 объектов. Но, как уже говорил, если объектов несколько тысяч, надо это все автоматизировать. К счастью, в Excel есть и такая функция. Функция называется vlookup.

Итак, что мы будем делать. Мы говорим компьютеру: найди нам цифру 1 в этом диапазоне ячеек, и потом выдай нам то, что написано в третьей колонке этого диапазона напротив этой единицы.

Не забудьте вставить клавишу F4, когда будете указывать диапазон ячеек, чтобы мы могли скопировать эту формулу на все остальные ранги.

В конце после указания диапазона вставляем слово False, чтобы нам выдали точное соответствие того, что мы ищем. Нажимаем Enter – отлично, нам нашли цветок с первым рангом и выдали вид этого цветка– это versicolor. Все верно.



Продублируем эту формулу для второго и третьего ранга. Ну вот и все, наша модель нашла, что ближайшие три соседа от нашего искомого цветка, все три соседа это цветки versicolor. И значит мы можем предположить, что с большой долей вероятности, наш искомый цветок тоже принадлежит виду versicolor.

Количество соседей, на основе которых мы принимаем решения, можно увеличить. Давайте посмотрим, кем являются 4 и 5 ближайшие соседи. Ок, получается 4 из 5 соседей – это versicolor и один сосед – virginica.

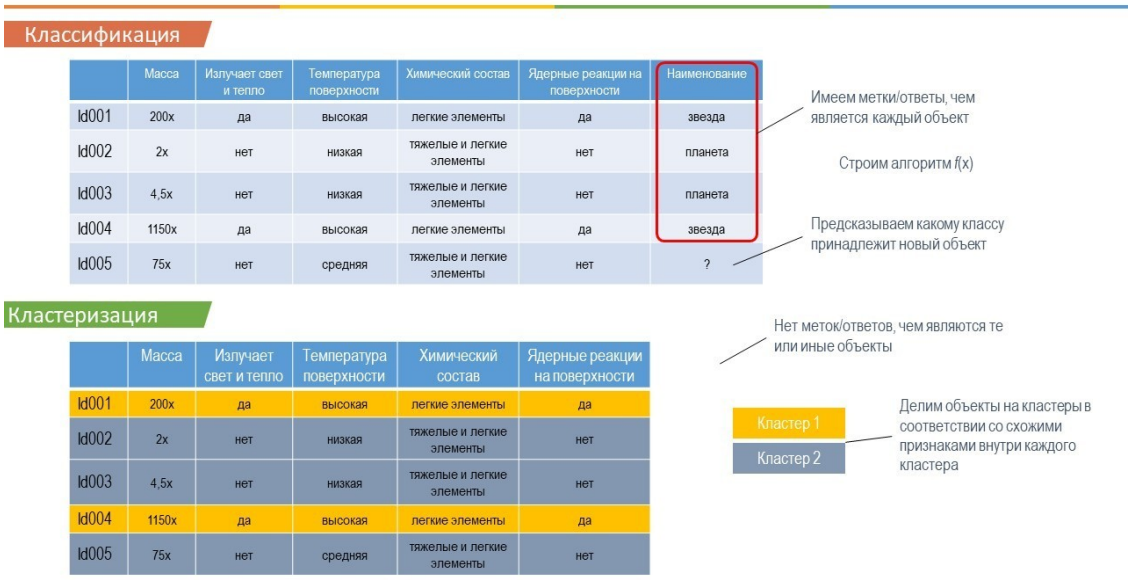
На данном простом примере мы объяснили, как работает метод К-ближайших соседей для решения задач классификации.

Данный пример был достаточно простой. Обычно, в выборке бывают тысячи объектов и не 2 признака как у нас было здесь, а намного больше. И решаются такие задачи не в Excel, а в специализированных программах, но суть решения таких задач, когда надо определить класс объекта такая же, как и в данном случае. Надеюсь, эта задача была полезной, попробуйте проделать то же самое самостоятельно, либо с этой же выборкой, либо попробуйте взять какие-либо другие данные.

Кластеризация

Кластеризация является одним из основных видов машинного обучения без учителя. В задачах кластеризации у нас имеется набор данных, и нам надо понять его внутреннюю структуру, а именно разделить эти данные на группы или кластеры таким образом, чтобы внутри каждого кластера находились элементы наиболее похожие между собой, и отличающиеся от элементов в других группах. У вас может возникнуть резонный вопрос. Чем кластеризация отличается от классификации. Давайте попробуем объяснить это на примере.

Предположим, у нас имеется таблица с признаками планет и звёзд, и рядом с каждым из небесных тел у нас сбоку указан ответ, чем оно является: планетой и звездой. Это ответы или так называемые метки, которые используются в обучении с учителем. На основе этих меток компьютер строит модель, по которой можно предсказать, чем будет являться новое небесное тело, которое нам попадется в следующий раз при наблюдении. Это пример классификации, когда мы предсказываем к какому классу относится новый объект, основываясь на наших помеченных данных. А вот если бы у нас не было ответа какой объект чем является, но у нас были бы только признаки, такие как масса, температура, состав и т.д., то мы могли бы просто поделить все имеющиеся объекты на 2 группы в зависимости от тех или иных признаков. Наш компьютер бы нам сказал, что вот в этом наборе объектов присутствует два очевидных кластера, которые отличаются между собой. Нам бы самим пришлось давать названия этим двум кластерам, потому что компьютер этого не делает в задачах кластеризации. Потому что кластеризация – это пример обучения без учителя, когда мы не имеем меток или ответов для каждого объекта в нашей выборке. Мы просто разделяем их на группы, мы не пытаемся ничего предсказать в отношении новых объектов, а разделяем на кластеры уже имеющиеся объекты.



В каких случаях могут использоваться задачи кластеризации? Очень активно кластеризацию используют ритейлеры и большие магазины, когда они хотят понять, из кого состоят их клиенты. Возьмем, например, крупный гипермаркет. Чтобы делать точечные рекламные акции для своих потребителей, ему необходимо будет разбить их по группам или кластерам. Например, потребителей можно разделить на следующие группы: семейный кластер, в котором клиенты покупают товары для дома и детей; спортсмены, которые часто берут спортивные товары

и спортивное питание; дачники и так далее. И в следующий раз, когда будет акция, например, на спортивные товары, гипермаркет отправит информацию об этой акции не всем подряд потребителям, а только тем, кто в прошлом уже покупали спортивные товары. Или, например, потребителям из семейного кластера можно дополнительно предлагать детские товары или специальное детское питание.

Кластеризация также очень активно используется в социальных сетях. Это делается в первую очередь для того, чтобы разбить пользователей на группы по интересам и потом предлагать им более подходящее и релевантное видео, картинки, новости, чтобы они проводили в приложении всё больше времени. Во-вторых, это делается для того, чтобы более эффективно настраивать таргетированную рекламу, чтобы она была более точечной и направленной именно на те группы пользователей, кто является целевой аудиторией рекламодателя.

Смартфоны тоже используют алгоритмы кластеризации, они могут распределять фотографии по разным папкам, например, в зависимости от даты снимка или GPS треков, где были сняты фотографии. В биологии кластеризация используется для того, чтобы распределять те или иные виды животных или растений в определенные группы в зависимости от общих характеристик, которые имеют те или иные их представители. Очень часто алгоритмы кластеризации используются и в генетических исследованиях, в частности, для аннотации геномов и эволюционной биологии.

Таким образом, в кластеризации мы ничего не предсказываем, а просто распределяем имеющиеся объекты в разные кластеры или группы в зависимости от сходных признаков и параметров. Это помогает нам понять внутреннюю структуру нашего набора данных и более эффективно работать с ним в дальнейшем.

Ансамблирование в машинном обучении

Предположим, что у нас имеется некий алгоритм, и он в принципе неплохо справляется со своими задачами. Но что, если мы хотим улучшить точность предсказаний наших алгоритмов.

Для этого будут использоваться дополнительные более продвинутые методы. В этой книге мы рассмотрим ансамбли, бэггинг и случайный лес.

Ансамбли

Под ансамблями подразумевается, что мы будем использовать совокупность из нескольких разных моделей для решения одной и той же задачи с целью улучшения точности предсказывания.

Использование ансамблей в машинном обучении строится на теореме Кондорсе о жюри присяжных, которая была опубликована еще в 18-м веке. В соответствии с этой теоремой, если у нас имеется жюри присяжных, и каждый из членов жюри имеет независимое от других мнение, то есть их ответы независимы, и вероятность принятия правильного решения каждого члена жюри больше 50%, то тогда вероятность принятия правильного решения всеми присяжными в целом будет стремиться к 100% по мере увеличения количества членов жюри. То же самое действует и в обратном случае. Если вероятность принятия ими правильного решения меньше 50%, то при увеличении количества присяжных, вероятность принятия правильного решения будет стремиться к нулю.

Еще одним показательным примером является так называемый пример «Мудрость толпы». В 1906 году в городе Плимут проходила ярмарка, на которой проходило в том числе и много разных увеселительных мероприятий. И на одном из таких мероприятий 800 человек приняли участие в конкурсе на отгадывание веса бычка. Ни один посетитель выставки не смог угадать точного веса быка, однако статистик Фрэнсис Гальтон высчитал, что среднее арифметическое всех предположений отличалось менее чем на 1% от реального веса быка. Бык весил 1207 фунтов, а среднее арифметическое получилось 1198 фунтов. То есть, даже если один человек не может дать правильный ответ, то если собрать данные от множества людей и усреднить их, то получается очень даже хороший и близкий к истинному результат.

Мудрость толпы применяется сегодня во многих отраслях. Например, помощь зала в игре «Кто хочет стать миллионером», и даже создание Википедии или Ответов на Yahoo, Mail.ru и других веб сайтах базируется на информации, которую предоставляют большое количество пользователей, и если посчитать, что большинство пользователей предоставляет правдивую информацию, то с каждым новым пользователем достоверность будет все повышаться и качество информации будет улучшаться.

Так вот, эту теорему Кондорсе о жюри присяжных и идею о мудрости толпы решили применить и в машинном обучении для улучшения точности алгоритмов.

Предположим, у нас есть несколько алгоритмов. Мы знаем, что один алгоритм ошибается на этом отрезке данных, а другой ошибается на другом отрезке данных, то при смешении результатов друг с другом, их совокупная ошибка будет падать, потому что они компенсируют друг друга. Таким образом, при использовании ансамблирования, совокупный результат нескольких моделей практически всегда лучше в том что касается точности предсказания, по сравнению с использованием только одной модели.

Представим еще один пример. Есть такая притча о группе слепых людей и слоне. В этой притче нескольким слепым людям дают потрогать слона. Кто-то трогает его уши, кто-то берет его хобот, кто-то дотрагивается до его хвоста или ног. Для каждого из этих людей слон будет иметь разное представление. Однако если объединить высказывания их всех, то получится очень даже полноценное представление о том, как реально выглядит настоящий слон. Это и есть пример ансамбля, когда с помощью разных моделей мы делаем точность нашего предсказания лучше.

Ансамбли могут строиться по-разному. Самые простые способы – это комитет большинства, выбор среднего и выбор взвешенного среднего.

Комитет большинства

Комитет большинства означает, что мы выбираем то значение из всех моделей, которое встречается наиболее чаще.

Возьмем следующий пример. Предположим, мы хотим купить новый телефон. Мы выбираем между американскими, китайскими и корейскими моделями. Если бы мы не использовали метод ансамбля, мы бы, например, пришли в магазин, спросили бы у продавца что он порекомендует, и взяли бы эту модель.

Если мы используем ансамбль, то перед выбором покупки, мы проведем сначала опрос у разных людей, у наших друзей, посмотрим отзывы пользователей в интернете, посмотрим обзоры моделей на Youtube и рекомендации экспертов.

И если мы выбираем способ «комитет большинства», то в таком случае мы выберем ту модель телефона, за которую проголосовало простое большинство людей.

Следующий способ – **выбор среднего**.

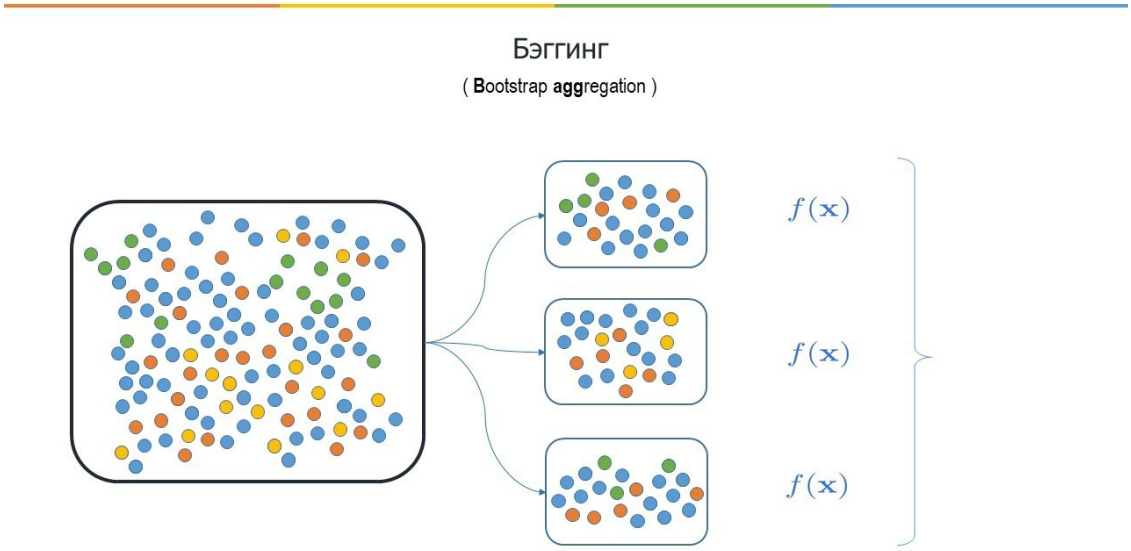
Предположим, у нас есть портал кинофильмов и каждому фильму мы присваиваем оценку от 1 до 10. Все пользователи, кто посмотрел фильм, ставят свою оценку, и далее мы выводим среднюю оценку на основе всех этих данных. Например, 2 человека поставило оценку 7, 6 человек поставило оценку 8, 12 человек оценку 9, и 9 человек поставили максимальную оценку. В итоге, получается средняя оценка – 8,96.

И следующий способ **– выбор взвешенного среднего**. Опять же возьмем пример с оцениванием кинофильмов. Здесь разным группам зрителей мы можем дать разные веса. Например, профессиональные кинокритики получают вес 1, пользователи, которые на платформе уже давно, получают вес 0,75, а пользователи, которые зарегистрировались только недавно, получают вес 0,5. Далее, мы выводим взвешенную среднюю из их оценок. И таким образом, получаем более качественную оценку, так как предполагается, что кинокритики и киноманы лучше разбираются в фильмах.

Немного более продвинутые ансамблевые методы классификации – это бэггинг, бустинг и случайный лес.

Бэггинг и бустинг

Бэггинг – это сокращенное от английского названия bootstrap aggregation. Предположим, у нас имеется большой набор данных или датасэт. Мы начинаем в произвольном порядке вытаскивать из этого набора объекты и делать из них более мелкие датасэты. В итоге мы получаем несколько новых датасэтов, которые по своей структуре будут сильно похожи на наш изначальный датасэт, но в то же время будут немного отличаться. Некоторые объекты будут встречаться в нескольких новых датасэтах, то есть они будут пересекаться, и это нормально. И далее мы будем обучать наши алгоритмы уже на этих новых датасэтах, таким образом улучшая нашу точность.



Предположим, мы – фармацевтическая компания, и у нас имеется данные о 10 000 пациентах и их реакции на новый препарат, который мы изобрели. Для некоторых из пациентов он подходит очень хорошо, для других он не действует вовсе. У нас имеется дерево решений, и обучив наш алгоритм мы получаем модель, которая выдает правильные решения в 75% случаев. Это конечно неплохо, но все-таки 25% ошибок – это достаточно большой разброс. Поэтому мы разбиваем наш датасэт о пациентах на несколько более мелких датасэтов, например, чтобы в каждом было по 2000 пациентов. И потом тренируем наши алгоритмы уже на каждом новом датасэте. И далее мы аггрегируем полученные алгоритмы и усредняем их в конечную модель, которая теперь будет выдавать правильные решения уже в более чем 80% случаях.

Бэггинг, как правило, используют когда высока дисперсия ошибки базового метода. Бэггинг также полезно использовать в случаях, когда изначальная выборка не такая большая, и в связи с этим мы создаем много случайных выборок из исходной выборки. Хотя элементы в таких подвыборках могут дублироваться, как правило, результаты после агрегации результатов оказываются точнее, чем если бы мы брали только изначальную выборку и обучали алгоритм по ней.

Бустинг – это тоже способ построения ансамбля алгоритмов, основанный на последовательной технике, когда каждый последующий алгоритм стремится компенсировать недостатки предыдущих алгоритмов.

Сначала первый алгоритм обучается на всем наборе данных, а каждый последующий алгоритм строится на примерах, в которых предыдущий алгоритм допустил ошибку, таким образом, наблюдениям, которые были предсказаны некорректно предыдущей моделью, дается больший вес.

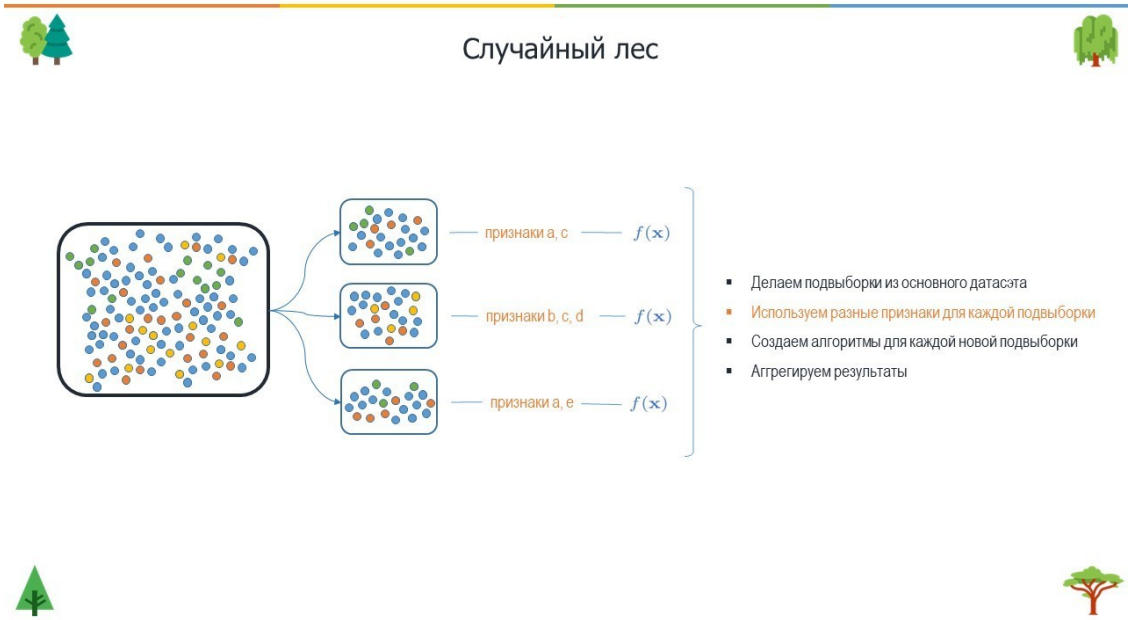
Бустинг в последние годы остается одним из самых популярных методов машинного обучения наряду с нейронными сетями. Его основные преимущества – простота, гибкость и универсальность. Одним из самых популярных примеров алгоритма бустинга является адаптивный алгоритм AdaBoost, разработанный Шапире и Фройндом еще в конце 90-х годов прошлого столетия.

Случайный лес

Алгоритмы случайного леса используются в задачах машинного обучения очень часто, причем они могут применяться в самом широком спектре задач – задачах кластеризации, регрессии и классификации.

Случайный лес строится примерно так же как и бэггинг, но это его немного усложненная версия. Сходство с бэггингом заключается в том, что мы из нашего большого набора данных или выборки, делаем несколько подвыборок.

Различие заключается в том, что в бэггинге при построении алгоритмов мы используем все признаки. А в случайном лесу мы случайным образом выбираем только несколько признаков, на основе которых мы будем строить каждое отдельное дерево.



У нас был пример с фармацевтической компанией, где нам надо было понять будет ли иметь какой-то эффект наш новый медицинский препарат на людей с разными признаками, в частности, возраст, пол, уровень холестерина и давление.

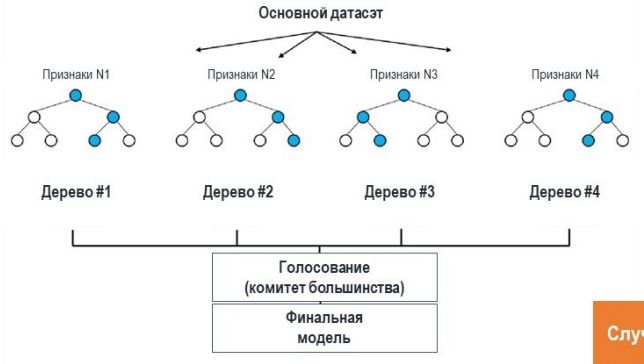
В бэггинге мы делали подвыборки и для каждой подвыборки строили дерево решений, основываясь на всех признаках. В случайном лесе, мы также будем делать подвыборки, но кроме этого, в каждой подвыборке мы будем случайным образом брать только отдельные признаки, например, только пол и возраст, или возраст и уровень холестерина.

Случайный лес

ID пациента	Возраст	Пол	Давление	Уровень холестерина	Эффективность препарата
001	16-28	Ж	Низкое	Нормальный	Положительно
002	29-45	Ж	Высокое	Высокий	Не замечено
003	46-64	М	Высокое	Нормальный	Не замечено
004	16-28	Ж	Среднее	Нормальный	Положительно
005	29-45	М	Низкое	Высокий	Не замечено
006	29-45	М	Низкое	Высокий	Положительно
007	46-64	Ж	Среднее	Нормальный	Положительно

Далее, как обычно мы будем агрегировать наши результаты и получим еще большую точность предсказаний наших алгоритмов. В задачах классификации нашу финальную модель мы будем выбирать комитетом большинства, а в задачах регрессии мы будем использовать среднее арифметическое.

Случайный лес



Точность:

выше чем у модели для основного датасэта

Случайный лес:

- Высокая точность прогнозирования
- Трудности в интерпретации результатов

Но у случайных лесов есть существенный недостаток. Интерпретация прогнозов становится очень трудной. Потому что у нас имеется много деревьев разной глубины, в каждом дереве используются разные сочетания признаков. И поэтому в случаях, когда нужна прозрачность принятия решений, случайный лес практически не используется. Например, при кредитном скоринге, когда банки принимают решение о выдаче кредита. В таком случае мы должны знать почему наша модель приняла решение о выдаче или нет, а при употреблении случайного леса, это становится непонятным.

Таким образом, случайный лес – это как черный ящик, он очень хорошо прогнозирует, но практически ничего не объясняет, то есть непонятно на какой основе строятся эти прогнозы. Однако, для задач, где нужна высокая точность предсказания, случайный лес – это один из основных алгоритмов.

Будущее Искусственного интеллекта

Предсказывать будущее – неблагодарное дело, и как правило, если заглядывать совсем далеко вперед, то все оказывается не так как мы ожидали, потому что появляются совсем другие технологии, о которых сейчас мы даже не знаем.

Поэтому хотелось бы сосредоточиться на ближнесрочной и среднесрочной перспективе развития ИИ, возьмем, к примеру, ближайшие 5-20 лет. И будем основывать свои прогнозы развития ИИ на том, что уже начинает происходить в этой области.

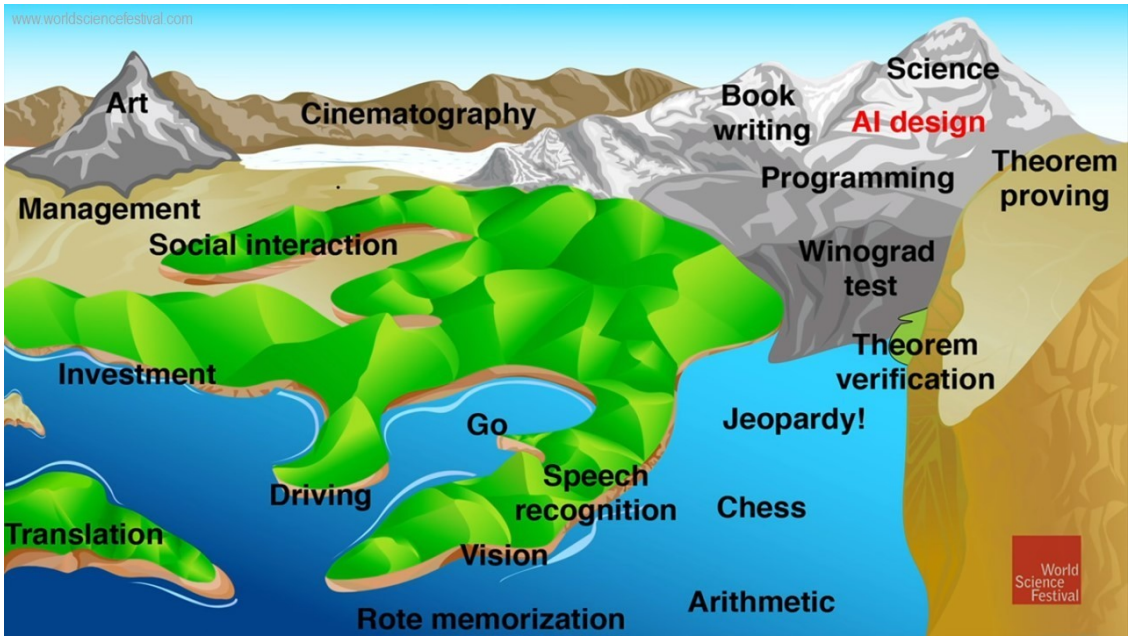
Во-первых, ИИ проникнет практически в каждую отрасль нашей жизни. Уже сегодня технологии машинного обучения и ИИ используются практически в любой отрасли, пусть и не повсеместно, но по крайней мере крупными компаниями в своей области или же наоборот стартапами, которые и задают новые тренды.

Помните, раньше сайты могли создавать только программисты, использую языки программирования HTML, CSS, Java и т.д. Сейчас же имеется большое количество конструкторов для обычных людей, где любой может создать свой сайт за несколько минут, используя различные шаблоны и нужную ему кастомизацию. И для этого совсем не надо владеть языками программирования.

То же самое будет и с ИИ. Будут созданы специальные автоматические конструкторы, которые помогут любому внедрить технологии ИИ, МО, нейронных сетей, не зная как программировать. Не надо будет знать Питон или другие языки, которые сейчас используются для этого. Достаточно будет сказать, что вам надо, и компьютер сам выберет подходящую модель и подберет необходимые данные. Естественно, это будет за определенную плату, потому что вам в любом случае нужен будет доступ к большим данным или вычислительным мощностям, что не может позволить индивидуальный человек. Хотя возможно крупные корпорации, такие как Гугл или Амазон или Фэйсбук будут предоставлять какие-то шаблонные простые решения бесплатно.

На самом деле такие решения, где не надо знать как программировать, уже присутствуют сейчас на рынке. Есть компании, которые предлагают шаблонные решения автоматизации бизнеса на основе ИИ и МО. То есть их покупают, как например покупают пакет Microsoft Office или подписку на Adobe. Но в дальнейшем это станет еще доступнее и возможно будет доступно и для физических лиц в более широком масштабе.

Далее, хотелось бы обратить внимание на данную картинку (источник: www.worldsciencefestival.com). Она отражает различные сферы, в которых ИИ уже показывает очень хорошие результаты и которые ему еще только предстоит покорить.



Те области, которые находятся под водой, это где ИИ уже работает лучше, чем человек. Например, компьютер уже может выиграть человека в шахматы, го, робоадвайзеры тоже зачастую показывают более хорошую доходность, чем финансовые консультанты. Следующая область, которая выделена зеленым, это то, что скоро также окажется под водой, то есть ИИ в ближайшее время улучшится и в этом. Это вождение автомобиля, компьютерное зрение и распознавание речи и социальное взаимодействие. Уже сейчас есть много чат-ботов и даже роботов, которые могут общаться с человеком, имитировать его эмоции, и недавно, компьютер даже прошел успешно тест Тьюринга, в котором люди, сидевшие за компьютером, посчитали, что они общаются с реальным человеком, а не с роботом. В ближайшем будущем технологии синтеза речи, и коммуникаций только продолжат развиваться, а значит нам будет еще сложнее понять с кем мы общаемся, с компьютером или человеком.

Как вы видите, самыми высокими областями на карте, до которых ИИ, еще далеко, являются области, требующие творческого подхода. Это написание книг и песен, программирование, кинопроизводство, и все, что связано с креативной деятельностью. Но здесь уже тоже есть подвижки. В интернете есть множество материалов, где вы можете найти музыку и стихотворения, написанные компьютером. И хотя в большинстве случаев понятно, что в них порой отсутствует логика или что-то не так, тем не менее творения ИИ очень быстро улучшаются с каждым годом.

Если говорить в глобальном масштабе, то многие аналитики считают, что мы сейчас переходим в совсем другую эпоху. Если вкратце, то известны четыре важные исторические эпохи, различающиеся по тому, как люди работали. Первая эпоха длилась несколько миллионов лет, и люди в этот период занимались охотой и собирательством. Дальше пришла сельскохозяйственная эпоха, когда люди научились жить на одном месте и заниматься выращиванием сельскохозяйственных культур. Эта эпоха продлилась несколько тысячелетий. Потом пришла индустриальная эпоха, когда мы научились создавать товары с помощью машин и мануфактур, и она продолжалась несколько столетий. И последняя Информационная эпоха продолжалась всего несколько десятилетий. И сейчас мы стоим на пороге новой эры, которую можно назвать Дополненной или Усиленной эпохой.

В эту эпоху наши человеческие способности расширяются, дополняются или усиляются с помощью компьютерных систем, которые позволяют нам более эффективно думать, роботизированных систем, которые позволяют нам создавать и делать большие вещи, и цифровых

нервных систем, которые позволяют нам выходить за пределы наших обычных человеческих чувств.

В будущем, если мы захотим что-то создать, нам не надо будет досконально разбираться в этой области. Мы просто обратимся к компьютеру и скажем, например, давай создадим дизайн нового гоночного автомобиля. И компьютер сам спроектирует его, учитывая все необходимые данные, протестирует в своей программе лучшие варианты и просто выдаст уже конечный результат, к которому мы сами бы возможно даже не смогли прийти.

Мы сможем распознавать эмоции человека: совсем недавно ученые из Китая с помощью нейронных сетей начали распознавать крики младенцев и понимать, что конкретно он хочет в тот или иной момент, когда плачет. Хочет ли он покушать или попить, или что-то болит или ему надо поменять подгузники.

Кроме того, скорее всего большинство вещей будут иметь сенсоры и будут подключены либо к какой-либо центральной системе либо к системе производителя, чтобы передавать информацию об окружающей среде или о том, как потребитель воспринимает этот данный продукт.

Например, здания смогут передавать конструкторам или дизайнерам и архитекторам реальную информацию о том, как чувствуют себя люди внутри этого здания, и что необходимо изменить в плане эргономики и обустройства внутреннего пространства. Машины, проезжая по дороге, смогут автоматически передавать информацию городским властям о пробоинах в дорожном полотне, чтобы их починили в кратчайшие сроки.

Если брать в целом, то наше будущее может оказаться и совсем другим, но хочется верить, что наше будущее благодаря ИИ будет более разнообразным, эффективным, будет больше взаимосвязей между людьми и компьютерными системами, и что мы сможем создавать свои собственные миры, которые будут лучше того, что мы имеем, при этом уважая и учитывая интересы других людей и природы вокруг нас.

Если у Вас есть свои мнения о будущем ИИ, пишите их в отзывах к этой книге. На этой главе теоретический курс по Искусственному интеллекту и машинному обучению закончен. В дальнейших главах мы изучим основы программирования на Python – одном из основных на данный момент языков программирования, которые используются для решения задач машинного обучения. Спасибо, что остаетесь с нами.

Основы программирования на Python

Для тех, кто знает основы Python

В ближайших нескольких главах мы изучим основные понятия и команды в Питоне. Поэтому для тех, кто уже знаком с этим языком программирования и не хочет терять время, можно пропустить следующие несколько глав и перейти сразу к построению различных моделей машинного обучения в Питоне в следующих главах.

Либо если хотите освежить знания, можете продолжать читать главы по порядку, чтобы вспомнить основы программирования на Python.

Установка Python. Anaconda и обзор библиотек для МО

Языков программирования достаточно много, но де-факто и можно сказать исторически стандартом в машинном обучении и анализе данных стал язык программирования Python. Под него собрано множество библиотек. Конечно, могут использоваться и другие языки, как например, R, Scala или C#, но мы остановимся на Питоне, потому что он, пожалуй, самый популярный и достаточно простой для изучения.

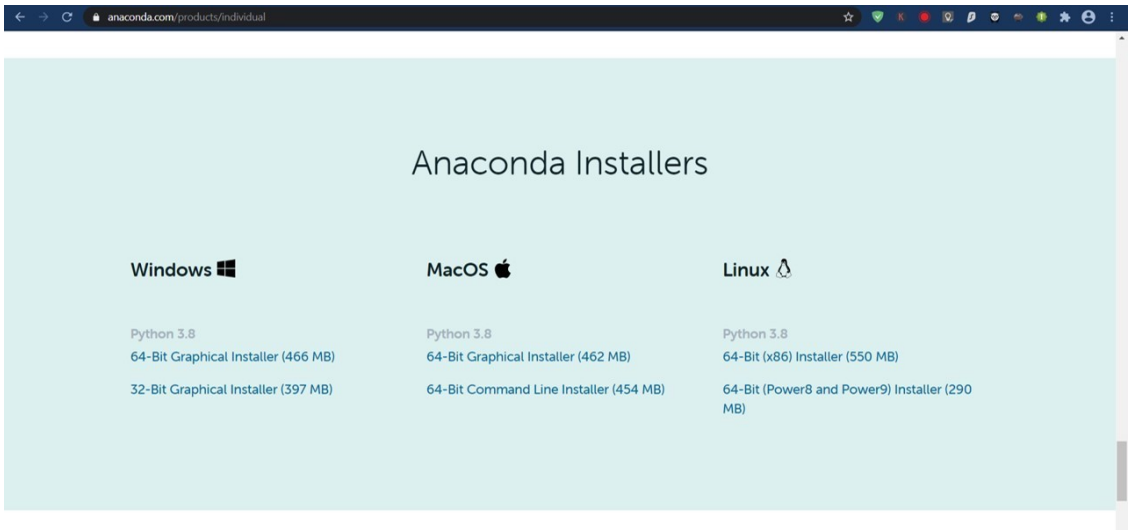
Есть такой замечательный сайт, называется Анаконда (www.anaconda.com). На нем можно скачать не только Питон, но и различные библиотеки под него, которыми пользуются дата саентисты, а также все те, кто занимается машинным обучением.

Что такое библиотеки? Библиотеки Python – это уже готовые решения, то есть модули с шаблонами кода. Их придумали для того, чтобы людям не приходилось каждый раз заново набирать один и тот же код: они просто открывают файл, вставляют свои данные и получают нужный результат.

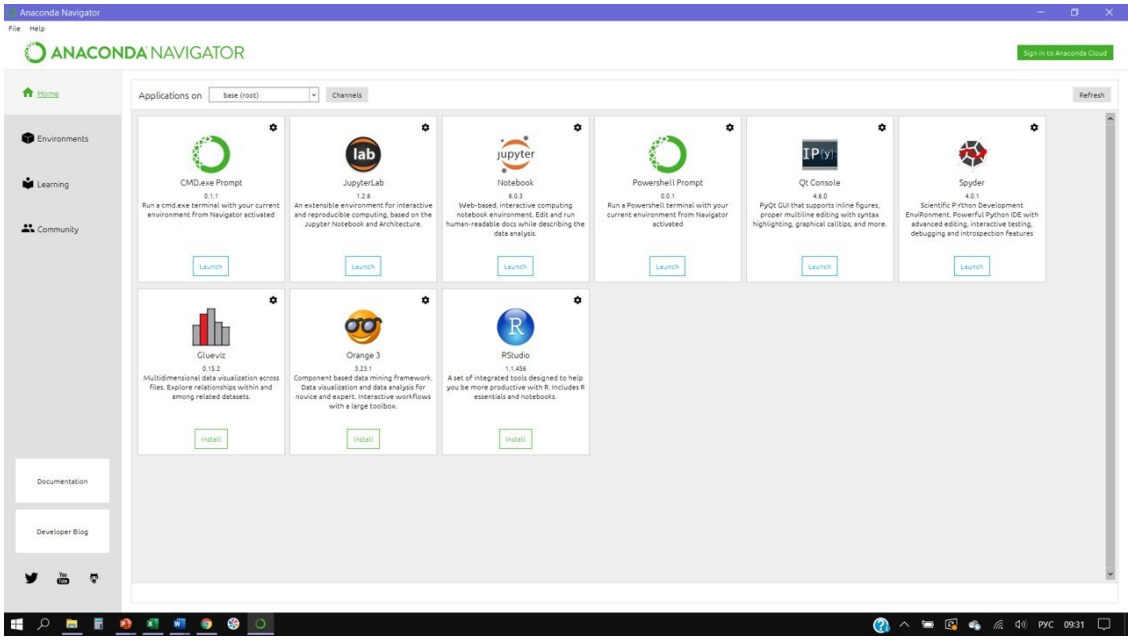
Итак, Анаконда – это опен-сорс дистрибутив, в который входят все основные инструменты для написания кода, анализа данных и различные библиотеки, которые могут помочь в этом процессе.

Первый инструмент – это Jupyter. Это среда, в которой мы будем писать код и тестировать его. Дальше различные библиотеки. Самая популярная – это Pandas. Это библиотека для работы с табличными данными. Она очень напоминает Excel. Вторая библиотека – matplotlib, она необходима для визуализации данных, создания диаграмм и графиков. Третья библиотека – это Scikit learn, в ней собраны основные алгоритмы машинного обучения. С помощью Scikit learn можно использовать такие алгоритмы как линейную регрессию, простые нейросети, дерево решений. И еще одна популярная библиотека – TensorFlow – она используется для работы с нейросетями различных архитектур.

Итак, давайте скачаем Питон, по умолчанию будет установлена последняя версия. На сайте Анаконды его можно скачать и для Windows, и для MacOs и для Linux.



После загрузки Анаконды у вас появится ярлычок Анаконда Навигатор, либо на Рабочем столе либо в меню Пуск, кликаем на него и откроется отдельное окно. Нас в первую очередь будет интересовать приложение Jupyter Notebook. И обратите внимание, что не стоит путать его с Jupyter Lab.



Кликаем на Jupyter Notebook и у нас откроется браузер. Юпитер позволяет создавать ноутбуки – это файлы с кодом Питона, которые можно тут же запускать и тестировать, создавать графики и т.д.

Давайте создадим папку и назовем ее. Теперь в этой папке создаем файл или так называемый ноутбук. И код будем писать в ячейках этого ноутбука. В следующих эпизодах мы начнем уже изучать основы программирования на Питон, самые базовые команды для начального ознакомления.

Если по каким-то причинам, вы не хотите устанавливать Питон, возможно, места не хватает на компьютере или еще по какой-либо другой причине, вы также можете воспользоваться сервисом Google Colab (https://colab.research.google.com/). Это среда, которую Гугл дает вам в аренду, и где вы также можете кодить на Питоне, не устанавливая его на свой компьютер. То есть это уже готовая система, на которой можно сразу начинать работать.

По такой же схеме работает среда Azure от Microsoft. Если у вас есть учетная запись в Microsoft, то можете пользоваться их тетрадками для Питона в их среде Azure без установки чего-либо на свой компьютер.

Введение в Питон. Базовые команды.

Итак, пару слов о Питоне. Это достаточно удобный язык для тех, кто только начинает учиться программированию. У него немного более простой синтаксис по сравнению с другими языками. В нем нет точек с запятой, команды, как мне кажется, выглядят проще, и не нужно постоянно следить, чтобы все скобки были открыты и закрыты, чтобы все модули были правильно уложены и т.д.

Давайте посмотрим на самые простые команды в Питоне.

Во-первых, как и во многих языках, если вы ставите решетку (**#**), то потом вы можете писать что угодно, это будет рассматриваться как комментарии, и они не будут воспроизводиться как код. Обычно эти комментарии нужны, чтобы вы сами понимали, зачем вы написали тот или иной код, когда будете его пересматривать, либо когда другие программисты будут на него смотреть, чтобы они понимали, почему вы написали код так, а не иначе.

 # комментарии

Дальше команда **print()** выводит то, что вы написали в скобках на экран. Если это текст, то его необходимо взять в кавычки, причем кавычки можно использовать как одинарные, так и двойные, разницы нет.

Print (“всем привет”)

Чтобы осуществить команду, нужно нажать Shift+Enter.

В Питоне, как и практически в любом языке программирования, можно выполнять любые математические операции, например, складывать. Например, печатаем 100+300, нажимаем Shift+Enter, получаем 400.

Мы также можем объявлять переменные.

a='Hello World!'
print(a)

Или мы можем сказать, что теперь буква **a** будет являться числом 100, а буква **b** – число 330.

И с помощью команды **print** вывести на экран их сумму, или произведение или любое другое математическое действие.

a = 230
b = 330
print(a + b)
print(a – b)
print(a * b)
print(a / b)

Числа также можно сравнивать между собой:

print(a < b)
print(a >= b) # a больше или равно
print(a == b) # a равно б
print(a != b) # a неравно б

Обратите внимание, если вы хотите объявить переменную вы ставите один знак равно, а если вы, например, сравниваете две переменные между собой, и спрашиваете равно ли одно другому, то тогда используем два знака равно подряд. А если мы хотим сравнить, что два числа неравны, мы ставим восклицательный знак и знак равно вместе.

Как и в любом языке программирования, в Питоне используются много разных типов переменных. Какие типы переменных являются самыми основными?

К неизменяемым типам (**immutable**) относятся: целые числа (**int**), числа с плавающей точкой (**float**), логические переменные (**bool**), то есть, например, True или False, и строки (**str**).

К изменяемым (**mutable**) типам относятся: списки (**list**), множества (**set**), и словари (**dict**). То есть по ходу дела мы можем их менять, вносить в них изменения и так далее.

Давайте посмотрим, как можно проверить к какому типу переменных относится та или иная переменная.

Мы можем использовать функцию **type()**, чтобы узнать класс переменной или значения.

```
a = 18
type (a)
int
a='Hello World!'
type (a)
str
a = 5.7
type (a)
float
a = True
type (a)
bool
```

Причем, как вы видите, в отличие от C# или Java, например, язык Питон характерен еще и тем, что, когда вы объявляете переменную, вам самому не надо изначально говорить, что за тип переменной вы сейчас объявите, Питон это сам понимает, вы просто пишите название переменной и потом даете ей значение. Поэтому Питон относят к языкам неявной сильной динамической типизации.

Если бы мы писали на языке C# или Java, мы бы указывали изначально к какому типу переменных он будет принадлежать.

```
Int (a) = 10;
```

А в Питоне мы просто указываем **a = 10** и Питон сам понимает, что в данном случае тип переменных является целым числом и сохраняет его таковым.

Что еще нам может понадобиться. Есть такое понятие, как форматированный вывод и работа со строками. Это очень удобная функция, которая позволяет нам не писать постоянно заново код, а как-то шаблонизировать его.

Например, чтобы не писать каждый раз имя, мы хотим, чтобы наша строка самостоятельно каждый раз подставляла нужное нам имя. Его можно оформлять по-разному, в зависимости от Питона, который у вас установлен. Вот основные варианты вывода.

```
name = "Timur"
print ("Hello, " + name)
Hello, Timur
```

Это самый типичный способ, когда мы просто складываем слово **Hello** и переменную **name**. Обратите внимание, наша фраза в кавычках, и не забудьте оставить пробел после запятой и перед второй кавычкой, а наша переменная **name** уже без кавычек.

Второй и третий вариант, это с использованием дополнительной буквы **f** или слова **format**, а также фигурных скобок.

```
print (f"Hello, {name}")
print ("Hello, {}".format(name))
```

Таким образом, если мы хотим изменить имя переменной, нам уже не нужно будет писать новую строку **print**, она автоматически подставит новое имя.

И еще одна самая базовая функция. Давайте посмотрим какую версию Питона мы используем.

```
import sys
print (sys.version)
```

И просто для интереса вы можете посмотреть какая версия Питона используется на Гугл Коллаб.

Итак, в этой лекции мы узнали:

– как выводить на печать,
– как выполнять математические действия и как сравнивать числа между собой,
– как объявлять разные переменные и к каким типам они могут относиться,
– как использовать форматированный ввод,
– и как узнать какую версию Питона мы сейчас используем.

If – оператор

Для базовых знаний в программировании на Питоне, нам также понадобится умение обращаться с так называемыми операторами для управления потоком команд. Это операторы **If** и **while**.

Давайте начнем с первого оператора **If**. Он используется для проверки условий. Если условие выполнено верно, то выполняется блок выражений, называемый **If** блок. Если условие неверно, то выполняется другой блок, называемый **Else** блок. Грубо говоря, мы говорим компьютеру, если случится одно условие, то выполни такую-то команду, а в противном случае, выполни другую команду.

Схема выглядит следующим образом:

```
If <condition>
<action>
elif <condition>
<action>
…
else:
<action>
```

Итак, давайте посмотрим, как выглядит оператор **If** на конкретном примере. Мы будем проверять наши числа на то, являются они четными или нечетными. Как мы знаем из школьного курса, четные числа – это такие числа, у которых остаток при делении на 2 является ноль, а у нечетных чисел остаток при делении на два является один.

В Питоне есть оператор процент, который показывает остаток, от деления одного числа на другое число. Например, если мы напишем **25%2**, то нам выдаст один, потому что если 25 поделить на два, то получится 12 и остаток 1. Если же написать **26%2**, то нам выдаст ноль, потому что 26 делится на два без остатка.

Итак, давайте теперь запишем формулу, которая будет выдавать нам ответ, является ли введенное число четным или нечетным.

```
number = 22 # введем любое число
if number % 2 == 0:
print("Число является четным")
else:
print("Число является нечетным")
```

Else в данном случае означает, что при всех других случаях, если остаток не равен нулю, то надо выдать ответ, что число нечетное.

Давайте проверим код на разных числах. Все работает.

Теперь если вы хотите, чтобы в вашем выводе присутствовало число, которое вы указали, вы можете также воспользоваться функцией форматирования. Выглядит она следующим образом.

Мы добавляем в строку print фигурные скобки после слова **number**, и после кавычек ставим точку, **format** и в круглых скобках пишем **number**. Таким образом в фигурные скобки будет автоматически ставиться то число, которое мы указываем в начале.

```
number = 22
if number % 2 == 0:
print("Число {} является четным".format(number))
else:
print("Число {} является нечетным".format(number))
```

Если у вас выходит ошибка, проверьте, что вы использовали именно фигурные скобки после слова **number**. А также поставили точку перед словом формат.

Попробуйте проверить другие числа, просто изменив вашу переменную **number**.

Давайте теперь добавим сюда же еще одно условие. Например, если мы ввели 0, многие спорят является ли 0 четным или нечетным числом. Давайте просто напишем, что при введении нуля, мы напишем “**Вы ввели 0**. **Введите, пожалуйста, другое число”**. Когда мы добавляем новое условие, мы используем оператор **Elif**.

```
number = 22
if number % 2 == 0:
print("Число {} является четным".format(number))
elif number == 0:
print("Вы ввели 0. Попробуйте другое число")
else:
print("Число {} является нечетным".format(number))
```

Но смотрите, что у нас выходит. Нам по-прежнему выдает, что ноль является четным. Все это потому, что Питон выполняет команды по порядку. То есть сначала он прочитал первую строку и поделил наше число на 2. А как мы знаем, при делении нуля на два, мы получаем ноль, то есть остаток тоже ноль. Поэтому он выдал команду, которая шла первой, что число является четным.

Как вы думаете, что можно сделать, чтобы избежать этого, и чтобы при делении на ноль, он выдавал нужную нам команду. Возьмите небольшую паузу и подумайте, можете даже самостоятельно попробовать в своем ноутбуке.

Итак, как вы, наверное, догадались, мы просто поменяем местами строки. Сначала первой строкой мы поставим проверку на то является ли введенное число нулем, а уже потом, проверку на то, какой остаток мы имеем.

```
number = 0
if number == 0:
print ("Вы ввели число 0. Попробуйте другое число.")
elif number % 2 == 0:
print ("Число {} является четным".format(number))
else:
print ("Число {} является нечетным".format(number))
```

Давайте попробуем еще одно задание, которое вы выполните самостоятельно. Напишите код, в котором мы указываем скорость движения автомобиля, и если скорость меньше или равна 60 км/ч, то мы пишем, **«будьте осторожны на дорогах»**, а если скорость больше 60 км/ч, то мы будем писать **«Пожалуйста, соблюдайте скоростной режим»**. Км/ч указывать в коде необязательно, просто число. И напомню, что для сравнения мы используем знаки **< , >,** меньше либо равно **<=**, и больше или равно **=>.**

А теперь можете попробовать выполнить это задание.

Ну как успехи, получилось ли у вас написать этот код и проверить его?

Код будет выглядеть следующим образом:

```
speed = 61
if speed <= 60:
print ("Будьте осторожны на дорогах")
if speed > 60:
print ("Пожалуйста, соблюдайте скоростной режим")
```

Здесь в принципе мы можем во второй строке использовать либо еще один оператор **if**, либо **else** – разницы особой нет. Давайте проверим разные скорости. И проверим если скорость равна 60. Как видим, все работает. Если хотите, можете самостоятельно попробовать придумы-

вать другие условия и попрактиковаться дома самостоятельно. Например, попробуйте написать код, можно ли продавать покупателю алкоголь или сигареты в зависимости от его возраста.

А в следующей главе мы продолжим знакомство с другими операторами.

Оператор While

Следующий оператор управления потоком команд – это оператор **while**. Он позволяет нам выполнять циклы задач, которые повторяются. **While** переводится как «пока / в то время как» и работает следующим образом.

Пока выполняется следующее условие, то необходимо выполнять такую-ту команду.

Самый простой пример, это когда мы просим компьютер вывести на экран все числа от 1 до 50.

Код будет выглядеть следующим образом. Сначала мы объявляем переменную **num**, и что она равна единице. Далее мы пишем **while**, пока наша переменная меньше или равна 50, то мы будем выводить на экран это число. С помощью команды **print**. И следующей строкой, мы будем с каждым циклом увеличивать это число на 1.

```
num = 1
while num <= 50:
print (num)
num += 1
```

Вот это написание **+=** как раз и обозначает, что мы даем нашей переменной новое значение с помощью прибавления к ней единицы.

Вот эти две строчки, которые написаны под **while**, **print** и увеличение переменной на единицу, являются одной итерацией или одним проходом цикла. Они выполняются только, если выражение **while** истинно. Внутри одного цикла может быть произвольное количество действий, и даже дополнительные циклы внутри основного цикла. **While** очень часто используется в компьютерных играх, при передаче данных, да и в принципе очень много где.

Как правило, внутри выражения **while** фигурирует переменная, которая явно или неявно меняет свое значение внутри цикла, потому что если она всегда будет одной и той же, то этот цикл будет постоянным и бесконечным.

Давайте попробуем сделать еще одно похожее задание. Попробуйте написать код с использованием оператора **while**, чтобы вам выдавали числа теперь в обратную сторону, от 50 до 0, но чтобы числа шли не по порядку, а через одну цифру, то есть 50, 48, 46 и так далее.

Возьмите паузу и попробуйте написать этот код самостоятельно.

Итак, я думаю вы справились и это было нетрудно. Код будет выглядеть следующим образом:

```
num = 50
while num >= 0:
print (num)
num -= 2
```

Теперь что еще нам надо изучить, когда мы говорим про оператор **while**? Во-первых, это как его останавливать, потому что циклы могут длиться вечно.

Мы уже говорили, что во-первых, его можно остановить, указав в самом начале ограничение, например **while** должно быть меньше 50. Но что если у нас не числа, а например, слова, или числа меняются не по порядку от нуля до 50, а в произвольном порядке. В таком случае нам может потребоваться команда **break**, которая останавливает цикл. Как это будет выглядеть. Возьмем тот же пример. Напечатать числа от 1 до 10.

```
num = 1
while True:
print (num)
num += 1
```

```
if num == 10:
break
```

В данном случае, если число равно 10, то используется команда **break**, которая обозначает, что мы прерываем цикл. И можно даже после того, как цикл будет остановлен, напечатать фразу, например, «сделано». Обратите внимание на отступы, потому что эта фраза должна уже быть написана на таком же уровне, как и изначальная фраза while, и тогда она будет выполняться только после того, как будет завершен цикл while.

```
while True:
print (num)
num += 1
if num == 10:
break
print ('Сделано!')
```

Итак, идем дальше. Теперь следующая функция, это как сделать так, чтобы компьютер спрашивал у нас что-то, а потом использовал то, что мы вводим для дальнейших операций. Для этого нам потребуется функция **input**.

Предположим, мы хотим сыграть в игру, которая называется «Угадай возраст». Мы задаем компьютеру возраст, который является истинным, и компьютер будет спрашивать у нас, чтобы мы его угадали. В зависимости от того, какой возраст мы укажем, компьютер будет либо продолжать просить у нас угадать возраст, либо скажет нам, что мы угадали. Как это можно реализовать в коде?

```
age = 10
while True:
if age == 30:
break
age = int(input('Угадай возраст '))
print ('Вы угадали')
```

Сначала задаем переменную **age**, ставим любое число, которое не равно нашему истинному возрасту. Далее пишем **while True**, это значит, мы даем команду, выполнять этот цикл вечно.

Потом оператор **If**, если возраст равен 30, и это тот возраст, который мы загадали в качестве верного, то тогда двоеточие и на следующей строке **break**, то есть мы прерываем цикл.

А если этот **if** не срабатывает, то тогда пишем **age** равно и далее следующая формула. **Int(input("Угадай возраст "))**. Эта формула означает, что теперь переменная **age** будет иметь такое значение, которое введет пользователь в ответ на фразу «**Угадай возраст**». То, что вначале стоит **int**, означает, что это будет числом. И дальше на следующей строке и уже на одном уровне со строкой **while** мы пишем **'Вы угадали'**. Эта фраза будет исполнена только когда прервется цикл **while**. Потому что, если возраст не будет угадан, компьютер будет прокручивать этот цикл и будет просить угадать возраст заново и заново.

Давайте протестируем. Как видим, все работает. Единственное, давайте вставим небольшой пробел после слова **Возраст**.

Еще такой момент, в принципе чтобы задать постоянный цикл, вместо слова **True** после **while** можно поставить любое правдивое утверждение, например пока 1<2, и компьютер будет выполнять этот цикл **while**, до тех пор, пока один меньше двух, то есть всегда. И поэтому принято в таких случаях писать просто **True**.

Итак, что мы прошли в этом эпизоде. Цикл **while**, как его можно останавливать, с помощью изначально заданного ограничения, либо с помощью команды **break**. Также мы прошли как захватывать и использовать данные, введенные пользователем, с помощью команды **input**.

Итак, теперь домашнее задание. Попробуйте по такому же принципу написать код, чтобы компьютер спрашивал у вас пароль до тех пор, пока вы не введете его правильно. И если вы вводите правильный пароль, то компьютер выдает фразу добро пожаловать. Небольшая подсказка: в строке, где вы будете захватывать введенный пользователем пароль, обратите внимание на тип переменной, будет ли у вас пароль числом или словом.

***

Пример ответа:

```
password = 'abc'
while True:
  if password == 'qwerty':
    break
  password = str(input('Введите пароль'))
print ('Добро пожаловать')
```

Строки

В этой главе мы разберем такой вид переменных как строки. Строки – это просто последовательность символов. Символы могут быть абсолютно разные: цифры, буквы кириллицей, латинские буквы, цифры и т.д.

Строки записываются через кавычки, причем можно записать как через одинарные, так и через двойные кавычки, никакой принципиальной разницы нет.

Со строками можно совершать стандартные операции. Их можно объединять через плюс:

‘foot’ + ‘ball’

‘football’

Можно умножать:

‘one’ * 3

‘oneoneone’

Причем, обратите внимание, если мы в строке, то есть в кавычках, дадим какое-то число, оно все равно будет рассматриваться как строка, то есть набор символов, а не как отдельное число. И поэтому если вы эту строку умножите, например, на три, у вас просто эти цифры встанут подряд три раза, а не умножатся.

‘100’ * 3

‘100100100’

Можно посмотреть длину строки через функцию **len**, которая показывает сколько символов в строке.

Len(‘football’)

8

Строки можно переводить на другую строку с помощью **\n**

print (‘first line’, ‘\n\n\n’, ‘Last line’)

Как мы уже говорили, строки являются последовательностями символов, и поэтому в принципе мы можем обращаться к отдельным символам в конкретной строке. Давайте посмотрим, как это можно делать.

Возьмем, например, переменную **Password** и она будет представлять собой строку, состоящую из символов **qwerty**.

password = ‘qwerty’

Символы в строках нумеруются, причем нумерация в строках идет с нуля.

Поэтому индекс ноль данной строки будет соответствовать букве **q**, индекс один будет соответствовать букве **w** и так далее.

```
password = 'qwerty'
i = 0
print (password[i])
q
```

Заметьте, что когда мы указываем индекс строки, мы используем квадратные скобки. И таким образом с помощью индексации мы можем обращаться к отдельным символам в данной строке.

Причем, мы можем также использовать и отрицательную индексацию. То есть если мы наберем индекс -1, то нам выдаст последний символ в строке, или первый символ с конца строки.

```
print (password[-1])
y
```

Помните, мы с вами говорили, что строки – это неизменяемые переменные, поэтому мы не можем просто взять и изменить какой-то один символ внутри строки.

Если мы захотим присвоить символу с индексом 2 внутри строки символ с буквой **а**, то у нас ничего не получится.

password[2] = 'a'

--

TypeError Traceback (most recent call last)

<ipython-input-13-857d85a5dd78> in <module>

--> 1 password[i] = 'a'

TypeError: 'str' object does not support item assignment

В таких случаях, если нам надо изменить какой-то отдельный символ в строке, нам надо будет просто заново задать нашей переменной эту всю строку с измененным в ней нужным нам фрагментом.

password = 'qwerty'

password = 'qwarty'

Мы также можем посчитать сколько раз встречается тот или иной символ в данной строке.

Предположим, у нас имеется строка, состоящая из символов **м** и **ж**. То есть, например, у нас имеется группа людей, и буква **м** обозначает мужчин, а буква **ж** соответственно женщин. Нам надо посчитать сколько в этой строке букв **м**.

Эту задачу можно решить двумя способами. Первый способ с помощью цикла **for**.

```
Group = 'мжжммжмжммжмжжм'
Cnt = 0
For i in group:
If I == 'м':
    Cnt+= 1
Print (cnt)
```

Сначала мы задаем счетчик, который для начала равен нулю. Потом мы задаем цикл, чтобы компьютер проверял каждую букву, и после проверки либо увеличивал счетчик, либо оставлял его неизменным.

Для каждого символа в группе, если символ равен **м**, то увеличить счетчик на 1. Вот эта комбинация плюса и равно (+=) означает, что теперь счетчик равен на одну цифру больше. И дальше выводим на печать наш счетчик.

Второй метод решения этой задачи немного проще.

Сначала мы выводим нашу строку.

Group = 'мжжммжмжммжмжжм'

И далее для этой строки мы вызываем метод **count**.

Print(group.count('м')

На этом со строками пока все, увидимся в следующей главе.

Списки

Итак, в этой лекции мы поговорим о таком элементе языка Python как списки.

Списки используются, чтобы хранить наборы значений, управлять ими, и изменять эти значения.

Элементами списка могут быть числа:

a = [3, 10, -4]

строки:

b = [‘Hello’, ‘How are you’, ‘Sasha’]

или даже другие списки:

c = [[3,4,10], [1,44, 23]]

Причем в рамках одного списка можно комбинировать объекты разных типов:

d = [32, ‘Hello’, 4.2, True]

Предположим, мы хотим сохранить температуру в мае месяце. Мы могли бы для этого создать 31 переменную и указать какая температура была каждый день. А если использовать список, то мы просто занесем температуру каждого дня в один список в квадратных скобках, и потом, если захотим, выбрать конкретный день.

Temperature = [21, 23, 23, 22, 24, 25, 28, 30, 29, 31, 27]

С помощью команды **len** мы можем узнать сколько объектов в нашем списке:

Family = [‘Anna’, ‘David’, ‘Mary’, ‘Sasha’]

Len(family)

4

И мы можем обратиться к каждому отдельному элементу списка с помощью индекса, как и в случае со строками. И не забудьте, что индексы нумеруются начиная с нуля, то есть, если мы напишем **family[2]**, то нам выдаст третий элемент по порядку в нашем списке.

Если мы хотим получить доступ к нескольким элементам из списка, то можем воспользоваться индексом и двоеточием:

family[0:2]

[‘Anna’, ‘David’]

Это будет означать, что мы берем элементы нашего списка с индексами от нуля до двух, причем первый индекс включительно, а второй указанный индекс не включается, то есть нам выдаст только элементы, которые имеют индекс ноль и один.

Если хотим взять элементы, но, чтобы они были построены в противоположном порядке, то в таком случае мы ставим два двоеточия и минус 1, то есть показываем, что хотим начать с последнего элемента в списке.

family[::-1]

Списки можно складывать или как это еще называется сцеплять.

Students = [‘Masha’, ‘Sasha’, ‘Maxim’]

Teachers = [‘John’, ‘Sylvia’]

Students + teachers

[‘Masha’, ‘Sasha’, ‘Maxim’, ‘John’, ‘Sylvia’]

К уже существующему списку мы можем добавить еще один элемент с помощью команды **append**.

family

family.append(‘Tim’)

family

[‘Anna’, ‘David’, ‘Mary’, ‘Sasha’, ‘Tim’]

Мы уже говорили, что в списках можно хранить элементы разных типов, например, создадим список **X** и поместим туда целое число, строку, логическую переменную True, число с плавающей точкой и давайте поместим туда еще один список.

X = [7, 'Hello', True, 6.3, [4, 7, 8, 3]]

Можно создавать пустые списки, в таком случае в квадратные скобки мы ничего не помещаем.

Мы можем также узнать есть ли тот или иной элемент в нашем списке. Например, давайте спросим есть ли в нашем списке **X** число 6.3. Это делается с помощью союза **in**.

6.3 in X

True

И Питон выдает нам ответ **True**, то есть это число есть в нашем списке. Давайте проверим, есть ли в нашем списке строка '**Hi'**

'Hi' in X

False

Как видим, Питон сообщил **False**, что значит, что такого элемента в нашем списке нет.

Если наш список состоит полностью из чисел, то с ним можно делать еще некоторые манипуляции. Давайте сделаем такой список исключительно из чисел.

N = [1, 3, 5, -33, 10, 23, 100]

С помощью команды **max(N)** мы можем найти максимальное число в этом списке, а с помощью команды **min(N)** – минимальное число.

Мы также можем суммировать все числа в нашем списке с помощью команды **sum(N).**

В принципе, вы можете даже найти среднеарифметическое какого-то списка. Предположим, у вас есть список из цен молока разных производителей.

Milk = [65, 120, 89, 143, 54]

Чтобы найти среднеарифметическое мы сумму этого списка поделим на его длину, то есть количество элементов.

Sum(milk) / len (milk)

Мы также можем отсортировать наш список с помощью функции **sorted (N).**

По умолчанию, эта команда сортирует по возрастанию, но, если вы хотите, можете отсортировать от большего к меньшего. Для этого надо будет добавить в скобках фразу **reverse = True**.

sorted (N, reverse = True)

Причем обратите внимание, после сортирования сам список и порядок, в котором расположены в нем элементы не меняется.

Если вы захотите выполнить все эти операции со списком, в котором у вас находятся не только числа, но еще и другие типы элементов, например, строки, то у вас ничего не получится, выдаст ошибку.

На этом со списками пока все. Потренируйтесь самостоятельно с созданием списков и выполнением различных операций с ними.

Словари

В этой лекции мы познакомимся с таким объектом языка Пайтон как словарь.

Итак, словарь в Пайтоне – это изменяемая неупорядоченная структура данных, которая позволяет хранить пары элементов «ключ» – «значение».

Как и в обыкновенном словаре мы можем, например, хранить переводы слов.

Abc = {‘apple’: ‘яблоко’, ‘water’: ‘вода’, ‘milk’: ‘молоко’}

Или мы можем создать словарь, чтобы сохранить в нем данные о том или ином пользователе. Например, его имя, рост, вес, город проживания и так далее. И потом вытаскивать либо всю информацию о нем, либо какие-то отдельные пункты.

Alex = {‘имя’:’Алекс’, ’возраст’:’25’, ‘вес’:’65’, ‘город’:’Казань’}

Как вы заметили, в словарях используются фигурные скобки, а пары элементов ключ – значение соединяются с помощью двоеточия.

Так как словари представляют собой изменяемый тип данных, то мы можем их изменять, добавлять или удалять из него элементы.

Можем изначально создать словарь пустым, а потом заполнить его.

D = {}

D[‘city’] = ‘Paris’

D

Обратите внимание, когда мы создаем либо берем и изменяем тот или иной **ключ**, мы берем его в квадратные скобки, а сам весь **словарь** находится в фигурных скобках.

Давайте теперь изменим название города.

D[‘city’] = ‘London’

D

А теперь давайте удалим эту пару и сделаем наш словарь пустым опять.

del d[‘city’]

D

Итак, как видим, мы можем изменять наш словарь, как пожелаем.

Почему мы сказали, что словари представляют собой "неупорядоченный" тип данных? Потому что в словарях в Питоне последовательность расположения пар не важна. Язык программирования ее не учитывает, и часто при выводе словаря последовательность вывода пар будет не совпадать с тем, как мы ее ввели изначально.

abc = {'cheese':'сыр','apple':'яблоко', 'water':'вода', 'tea':'чай','milk':'молоко'}

abc

И так как эти пары элементов внутри словаря не упорядочены явным образом, обращение к нужным нам элементам по индексам невозможно, можно только обращение по ключам и делается это через квадратные скобки по аналогии с индексами списков.

abc['apple']

abc['water']

Как уже говорили, если мы хотим добавить новую пару элементов в словаре, то делается это следующим образом. Название нашего словаря, далее в квадратных скобках название ключа, потом равно и значение этого ключа.

Abc = {‘apple’: ‘яблоко’, ‘water’: ‘вода’, ‘milk’: ‘молоко’}

Abc[‘lemon’] = ‘лимон’

Abc

Причем, если мы хотим изменить значение того или иного ключа, то здесь такой же синтаксис.

Alex = {‘имя’:’Алекс’, ’возраст’:’25’, ‘вес’:’65’, ‘город’:’Казань’}

Alex[‘вес’] = 67
Alex

Если вы хотите удалить элемент, то для этого используется оператор **del**.

del Alex[‘город’]
Alex

Можно удалить все элементы внутри словаря и оставить словарь пустым. Это делается с помощью метода **clear**.

Alex.clear()
Alex

Иногда возникает необходимость скопировать словарь полностью и в уже новый словарь добавить какие-то изменения. Скопировать словарь можно с помощью метода **copy**.

Alexbrother = Alex.copy()
Alexbrother
Alexbrother[‘город’] = ‘Москва’
Alex
Alexbrother

Мы с вами уже проходили списки, в которых мы можем хранить различные элементы. Так вот из списков можно создавать словари.

Предположим, у нас есть список с номерами от одного до пяти.

X = [1,2,3,4,5]

С помощью метода **fromkeys()** мы можем создать из этого списка словарь. Цифры от 1 до 5 в таком случае становятся ключами, которым пока не присвоены какие-либо значения.

Y = dict.fromkeys(X)
Y
{1: None, 2: None, 3: None, 4:None, 5: None}

Если мы хотим сразу присвоить какие-то конкретные значения, можно в этом же методе это указать.

Z = dict.fromkeys(X, ‘student’)
Z
{1: 'student', 2: 'student', 3: 'student', 4: 'student', 5: 'student'}

Если мы хотим, чтобы нам распечатали все ключи в словаре, можно воспользоваться циклом **for**.

Alex
for i in Alex:
print(i)

А если хотим, чтобы распечатались значения, то немного изменим строку print.

for i in alex:
print(alex[i])

Либо это можно сделать еще проще, просто указав, что мы хотим распечатать сначала ключи:

print (alex.keys())

а потом отдельно значения:

print (alex.values())

Мы можем даже распечатать фразы с использованием ключей и значений. Для этого нам надо сначала создать **кортежи**. Кортежи – это примерно то же самое, что и списки, только их нельзя изменять. Они заключаются в круглые скобки.

Возьмем, например, словарь, в котором содержатся имена людей и названия видов спорта, которые они любят.

Нам надо напечатать фразы, что они любят тот или иной вид спорта, то есть добавить слово «любит».

Для этого с помощью метода **items()**, сначала создаем особую структуру, состоящую из кортежей. Каждый кортеж будет включать ключ и значение.

sport = {‘Настя’:’теннис’, ‘Вова’:’хоккей’, ‘Саша’:’футбол’}

s = sport.items()

s

dict_items([('Настя', 'теннис'), ('Вова', 'хоккей'), ('Саша', 'футбол')])

Вот видите, теперь каждая пара заключена в круглые скобки, то есть они находятся в кортеже.

И дальше с помощью цикла **for** мы распаковываем кортежи и добавляем в области печати еще слово «любит».

for key, value in sport.items():

print(key, 'любит', value)

Итак, небольшое **самостоятельное задание** по итогам этой главы по словарям.

Предположим, у нас имеется компания с разными департаментами в ней: HR, Marketing, Sales, Administrative.

– Создайте словарь и наполните его данными по количеству людей в каждом департаменте, можете выбрать произвольные числа.

– Далее внесите изменения в департамент HR и удалите из него 2 сотрудников.

– Далее сформируйте новый департамент под названием – Finance.

– Потом удалите департамент Administrative.

– И, наконец, вычислите общее количество сотрудников в компании.

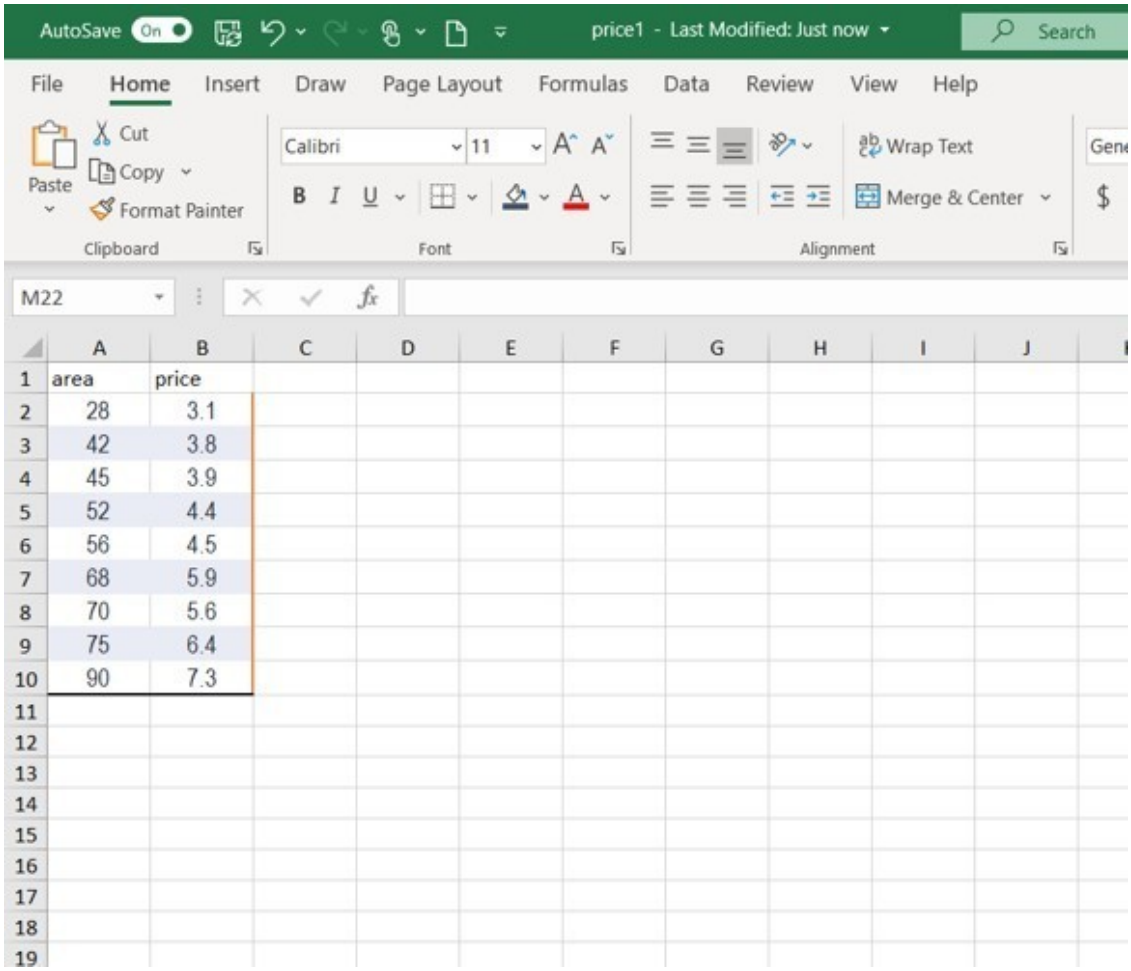
Построение моделей машинного обучения в Python

Предсказание цен на квартиры с помощью метода линейной регрессии

В этой главе мы научимся предсказывать цены на квартиры, используя Питон и алгоритм простой линейной регрессии. Помните, мы то же самое делали с помощью Excel?

Мы будем использовать тот же самый файл, и посмотрим, как это выглядит в Питоне. Потом в конце главы, я предложу вам выполнить самостоятельное задание.

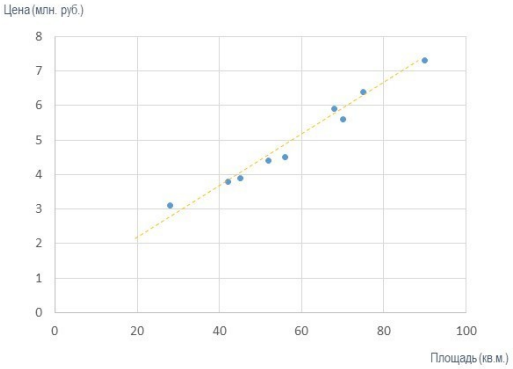
Итак, для начала можете скачать файл с ценами на квартиры по этой ссылке (https://github.com/timurkazantseff/machine-learning-101/). Вот перед нами файл в Excel, который указывает нам цену квартиры в зависимости от ее площади. Как вы понимаете, это очень упрощенная зависимость. Мы все понимаем, что на цены на квартиры влияет очень большое количество факторов помимо площади, а именно: город, год постройки дома, удаленность от центра города и от метро, состояние, этажность и так далее. Но для простоты обучения мы берем только один фактор – это площадь, потому что в принципе он является одним из самых основных, и зависимость тут очень сильная.



По сути, когда мы делаем такие предсказания, где есть линейная зависимость, все что нам надо сделать, это расположить наши значения на графике и провести между ними линию, которая бы максимально точно отражала зависимость. И таким образом, имея такую линию мы смогли бы предсказать какая будет цена в зависимости от любой площади.

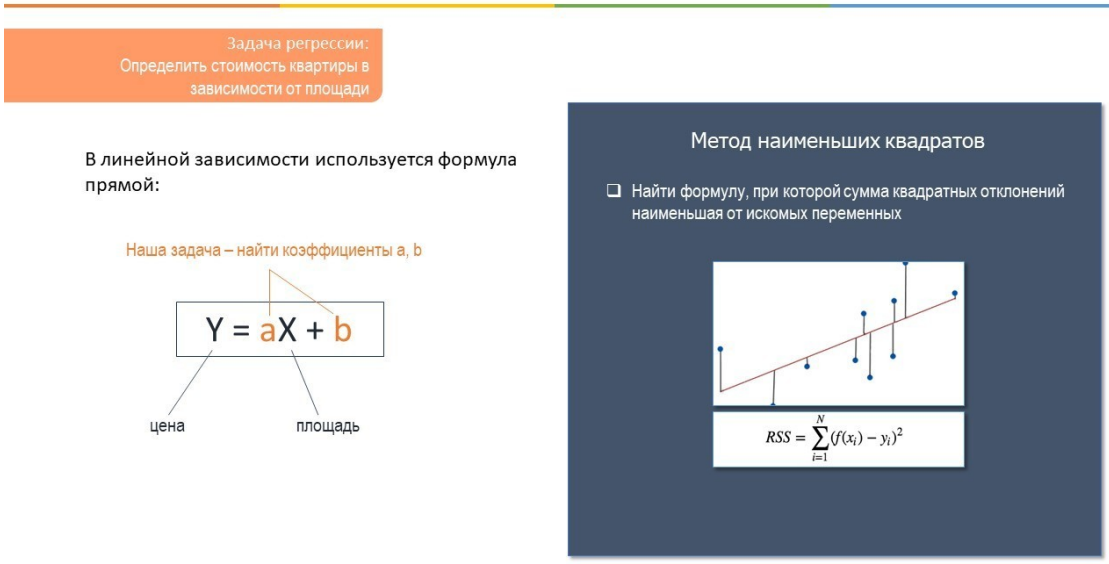
Задача регрессии:
Определить стоимость квартиры в зависимости от площади

Знаем	
Площадь (кв.м.)	Цена (млн. руб.)
28	3,1
42	3,8
45	3,9
52	4,4
56	4,5
68	5,9
70	5,6
75	6,4
90	7,3
Находим формулу зависимости $f(x)$	
Предсказываем	
34	?
49	?



Но вы можете спросить почему линия выглядит именно таким образом, ведь она не проходит через все точки.

Как мы уже говорили на лекции про регрессии, чтобы найти линию, которая максимально точно опишет нашу зависимость, мы будем использовать формулу прямой **Y = ax + b**, в которой в нашем случае **Y** является ценой, а **X** – площадью. И чтобы найти коэффициенты, мы будем использовать самый классический метод – это метод наименьших квадратов. **Метод наименьших квадратов** заключается в том, что вы находите такую формулу, при которой сумма квадратных отклонений наших имеющихся значений является наименьшей от линии, которую мы строим.



Давайте посмотрим, как это может выглядеть в Питоне. Сначала мы обучим нашу модель с помощью данных, которые у нас имеются, и потом мы сделаем предсказания для других квартир с другой площадью.

Помните, когда мы устанавливали Питон на сайте Анаконды, мы говорили, что Анаконда также содержит различные библиотеки для машинного обучения. Это были Pandas, Matplotlib, Numpy и другие. Эти библиотеки содержат уже готовые модули кода для различных алгоритмов машинного обучения.

Поэтому первое что мы делаем, это импортируем эти библиотеки в наш ноутбук в Юпитере.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
```

Pandas необходима для работы с табличными данными, как в Excel. Numpy облегчает математические операции в Питоне, а Matplotlib нужен будет для визуализации наших данных на графиках и диаграммах. Ну и еще одной важной библиотекой является Scikit-learn, в ней собраны основные алгоритмы машинного обучения. Из нее мы импортируем алгоритм линейной регрессии.

Далее, что нам надо сделать, это загрузить наш файл Excel в наш юпитеровский ноутбук, и чтобы он показывался как табличка, нам как раз пригодится библиотека Pandas, которую мы уже импортировали.

Сначала мы должны загрузить этот файл в ту папку, где находится наш юпитеровский ноутбук. Нажимаем upload, и загружаем файл в эту папку.

Теперь загружаем эту экселевскую таблицу в наш ноутбук, с помощью следующего кода:

```
df = pd.read_excel('price1.xlsx')
```

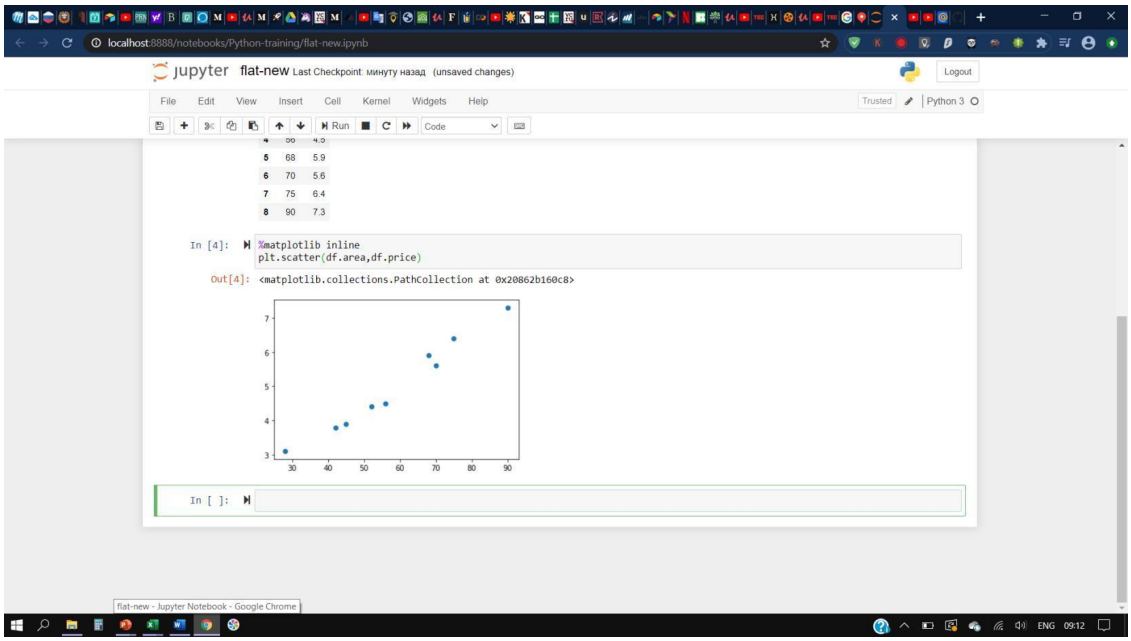
и с помощью команды **df** мы выводим эту таблицу на экран.

Если у вас файл с расширение **.csv**, что тоже часто встречается при работе с таблицами, то тогда вам надо будет использовать код:

```
df = pd.read_csv("price1.csv")
```

Теперь давайте расположим данные из этой таблицы на графике. Для этого нам потребуется библиотека **Matplotlib**, которую мы тоже вначале импортировали.

Итак, команда будет:
%matplotlib inline
plt.scatter(df.area,df.price)
В **plt.scatter** в скобках сначала мы указываем ось X, а потом ось У.



Отлично, мы видим наши значения на графике. Мы можем, кстати, сделать точки другого цвета, добавив в функции **scatter**, слово **color** и нужный нам цвет в кавычках.

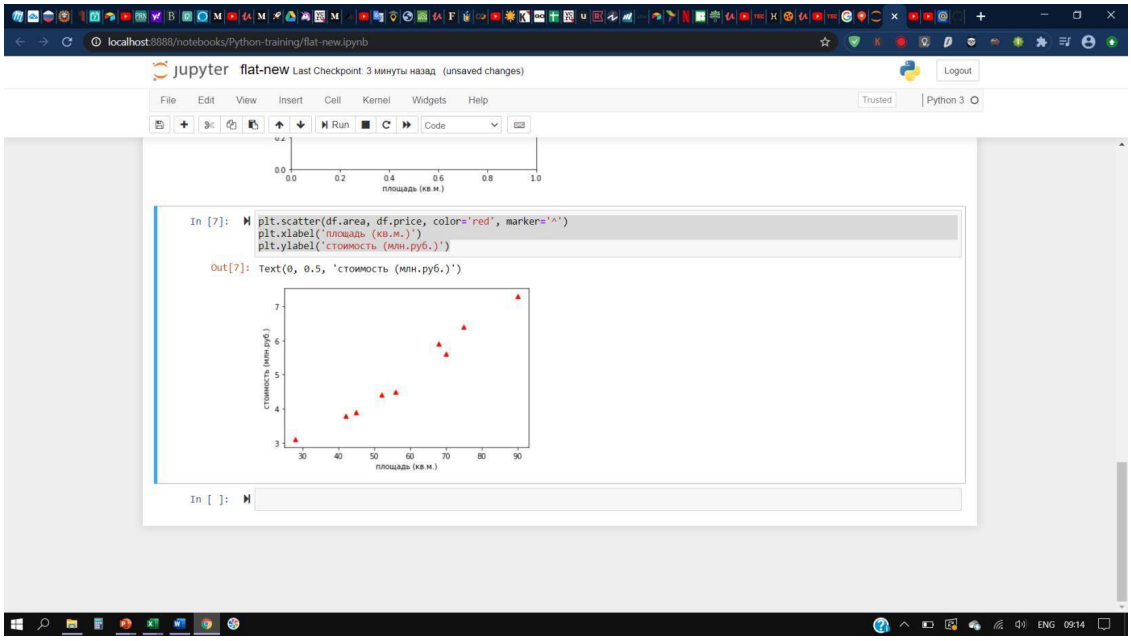
Мы также, если хотим, можем изменить точки на другие значки, например, плюсики, или звездочки, или вот такие стрелки. Для этого используем слово **marker** в функции **scatter**.

plt.scatter(df.area, df.price, color='red', marker='^')

Что еще нам не хватает? Давайте дадим нашим осям икс и игрек названия.

plt.scatter(df.area, df.price, color='red', marker='^')
plt.xlabel('площадь (кв.м.)')
plt.ylabel('стоимость (млн.руб.)')

Отлично, теперь выглядит как настоящий график.



Ну и теперь приступаем к тренировке нашей модели. Мы уже импортировали из модуля Scikit-learn шаблон линейной модели.

Давайте создадим переменную **reg**, и она будет являться нашей моделью линейной регрессии.

reg = linear_model.LinearRegression()

Обратите внимание на заглавные буквы в слове LinearRegression. Если вы введете прописные буквы, код не сработает.

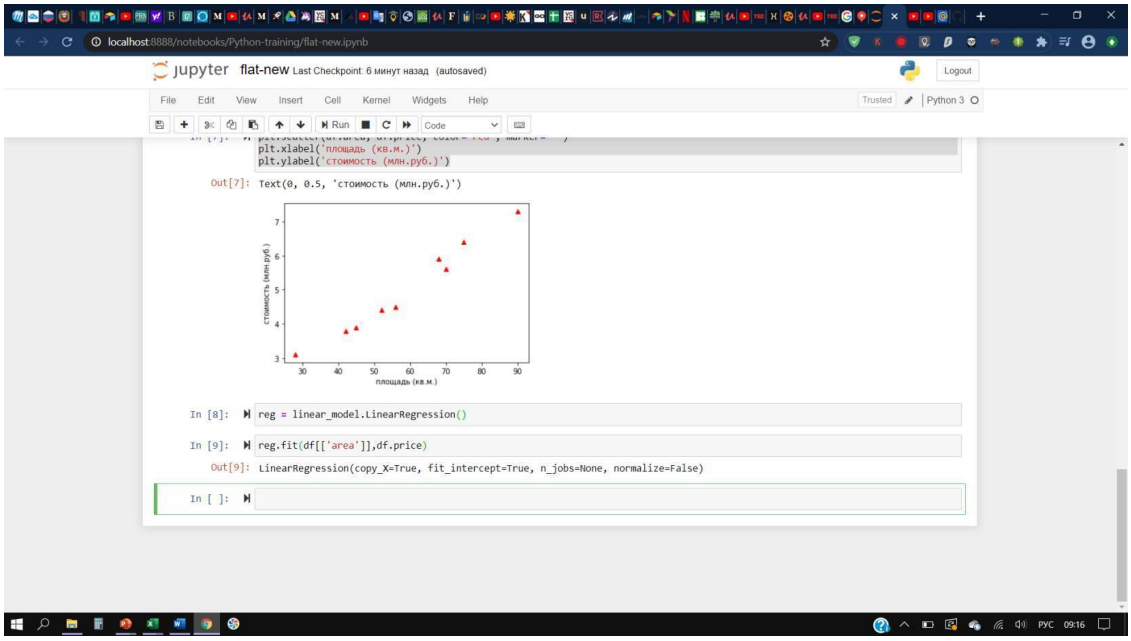
Дальше эту модель нам надо натренировать с помощью имеющихся у нас данных, то есть нашей таблицы. Грубо говоря, нам надо ей показать, что вот есть такая-то площадь и по ней цена такая то, есть другая площадь, и по ней цена такая. Пожалуйста, с помощью этих данных вычисли наилучшую формулу линейной регрессии, которая описывает такую зависимость цены от площади.

Итак, загружаем в нашу модель наши данные по площади и ценам.

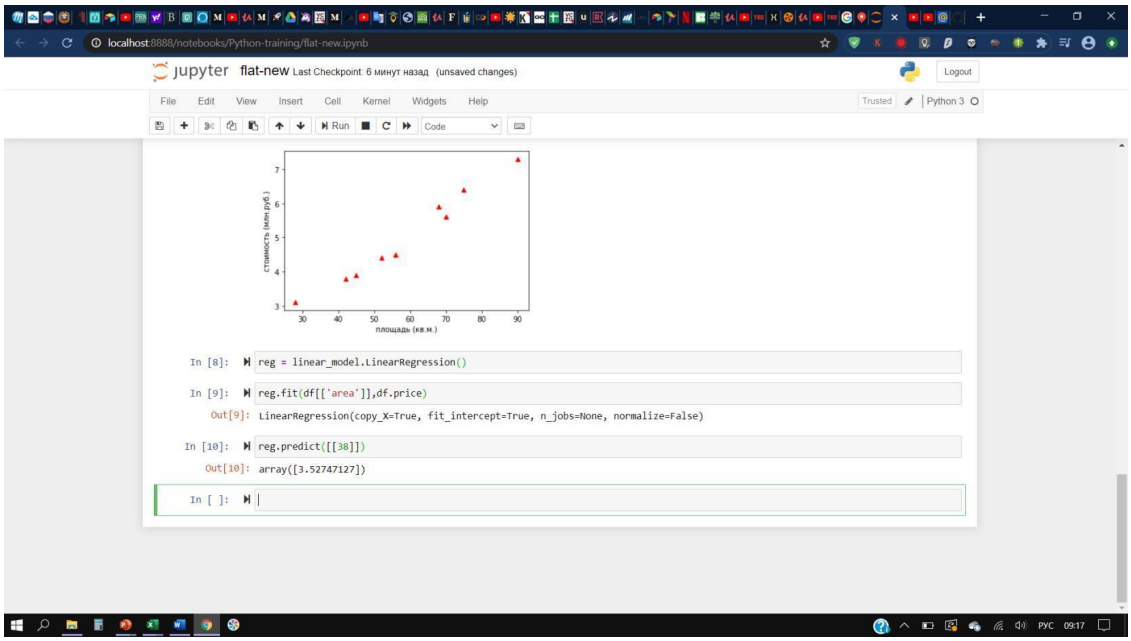
reg.fit(df[['area']],df.price)

В двойных квадратных скобках мы даем значения наших факторов, в нашем случае площади, а после запятой, ответы, в нашем случае цены. На самом деле факторов или колонок с факторами может быть несколько, и тогда мы бы записали их через запятую.

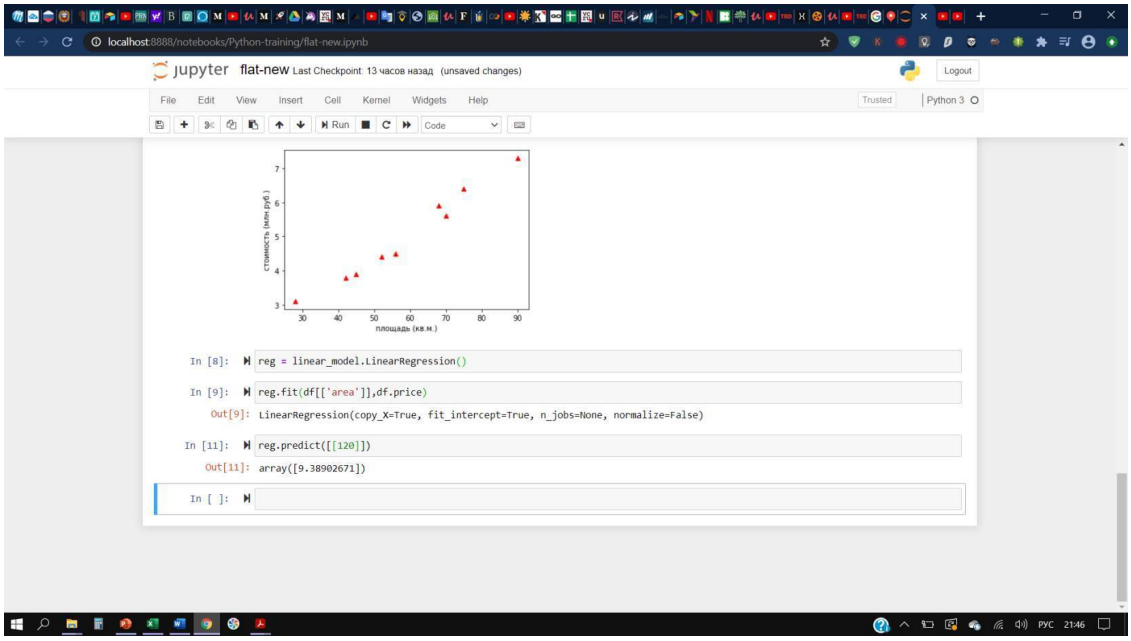
Все нам выдало подтверждение, что модель обучилась, и теперь она может предсказывать.



Давайте попробуем предсказать сколько будет стоить квартира из 38 кв.м.
reg.predict([[38]])



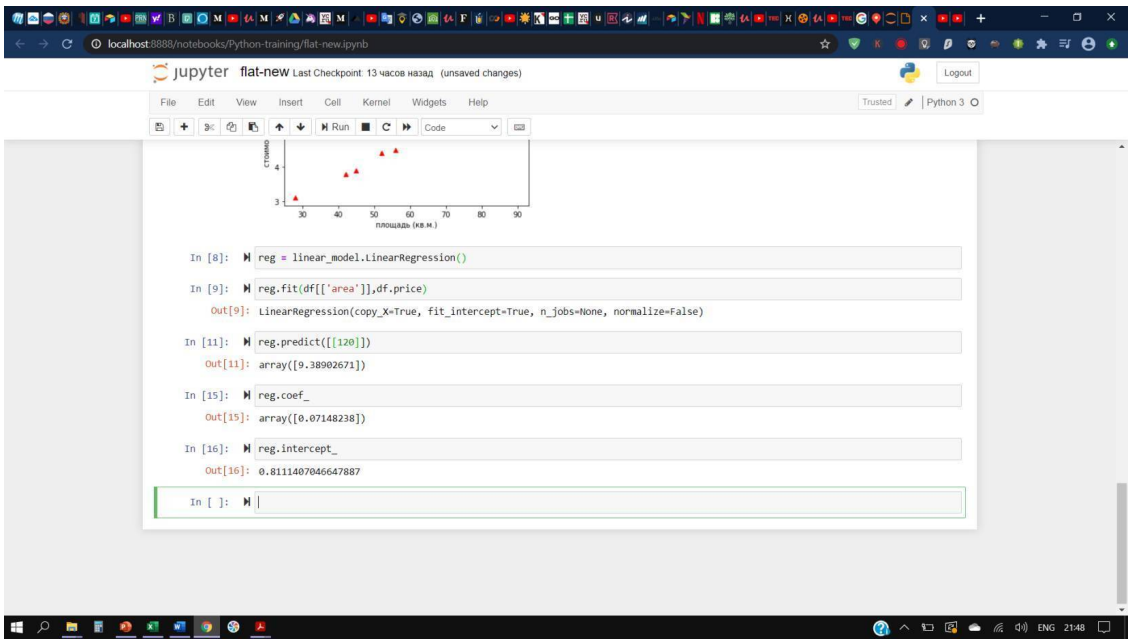
Отлично, давайте посмотрим сколько будет стоить квартира площадью 120 метров.



Давайте сравним с нашими предсказаниями, которые делались в Excel. Они очень похожи с точностью до нескольких сотых.

Итак, мы говорили, что вся линейная регрессия строится по формуле **Y = ax + b**. И наша задача, задача алгоритма линейной регрессии, найти такие коэффициенты **a** и **b**, которые дадут минимальное отклонение наших величин от линии, которую мы строим.

Давайте посмотрим какие коэффициенты наша модель получила. Для коэффициента **a** мы пишем **reg.coef_**, а для коэффициента **b** – мы пишем **reg.intercept_**.



Таким образом наша модель выглядит следующим образом:

Стоимость квартиры = 0.07148238 умножить на площадь и плюс 0.8111407046647887.

Давайте сверим, что нам давал Excel.

y = 0,0715x + 0,8111

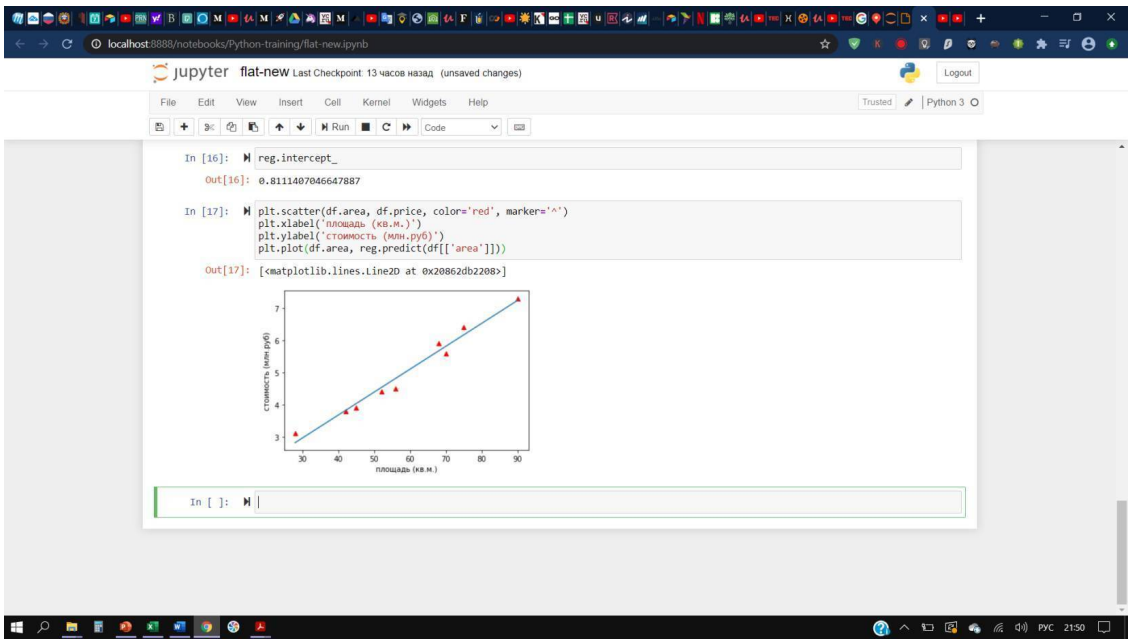
Как видим, очень похожие результаты, просто в Excel они немного округленные.

Давайте проверим еще раз.

Подставим эти коэффициенты под нашу формулу, умножим 120 метров на первый коэффициент и прибавим к нему второй коэффициент, и да – мы получили как раз то число, которое нам выдало в предсказании выше.

Кстати, мы можем начертить линию, которая показывает наши предсказания. Мы использовали алгоритм линейной регрессии, поэтому как раз у нас будет и прямая линия.

Чтобы увидеть ее на нашем графике, давайте заново скопируем код, который мы использовали выше для получения графика. И в последней строчке добавим **plt.plot** и в скобках **df.area**, **reg.predict(df[['area']])**.



Этот код означает, начертить линию, у которой по оси икс мы будем иметь площадь, а по оси игрек у нас будут цены, которые предсказала наша модель. Обратите внимание на количество квадратных и круглых скобок, не ошибитесь, когда будете вводить этот код.

```
%matplotlib inline
plt.scatter(df.area, df.price, color = 'green', marker = '*')
plt.xlabel('площадь (кв.м.)')
plt.ylabel('стоимость (млн.руб.)')
plt.plot(df.area, reg.predict(df[['area']]))
```

Отлично, как видим, наша линия достаточно хорошо описывает зависимость цены от площади, есть небольшие отклонения от истинных значений, но они неизбежно будут, но, по крайней мере, мы точно можем сказать, что это лучший вариант из всех возможных прямых линий для данных точек.

***

Итак, давайте продолжим и предположим, что у нас есть файл с квартирами, у которых мы знаем их площади, но не знаем цены. Нам надо будет заполнить вторую колонку с помощью цен, которые предскажет наша модель.

Давайте сначала загрузим этот файл в наш юпитеровский ноутбук (файл prediction_price.xlsx можно скачать по ссылке https://github.com/timurkazantseff/machine-learning-101). Для этого этот файл, во-первых, должен находиться в той же папке, где и ваш

юпитеровский ноутбук. Теперь используем команду **read_excel**, чтобы занести файл в наш ноутбук.

pred = pd.read_excel ('prediction_price.xlsx')

Далее, пишем **pred**, чтобы вывести нашу таблицу на экран.

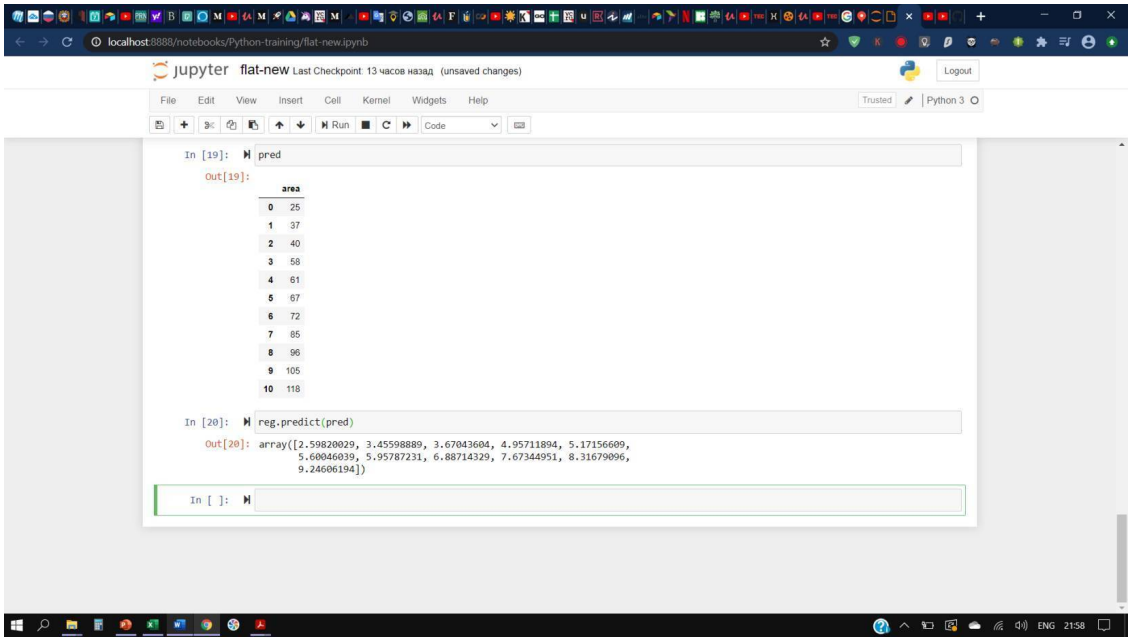
Здесь 10 позиций. Когда же файлы слишком большие, чтобы они не мешались на экране, можно просто попросить компьютер вывести самые первые строчки с помощью команды head.

pred.head()

Либо мы можем даже указать сколько строчек таблицы мы хотим вывести.

pred.head(3)

Далее мы используем нашу уже имеющуюся модель, чтобы сделать предсказания по этим новым площадям квартир. В формулу **reg.predict** мы подаем наши данные по площади, и нам выдает уже готовые ответы с ценами.



Отлично. Теперь наша задача сделать в этой таблице еще одну колонку, чтобы мы могли туда вставить эти предсказанные цены.

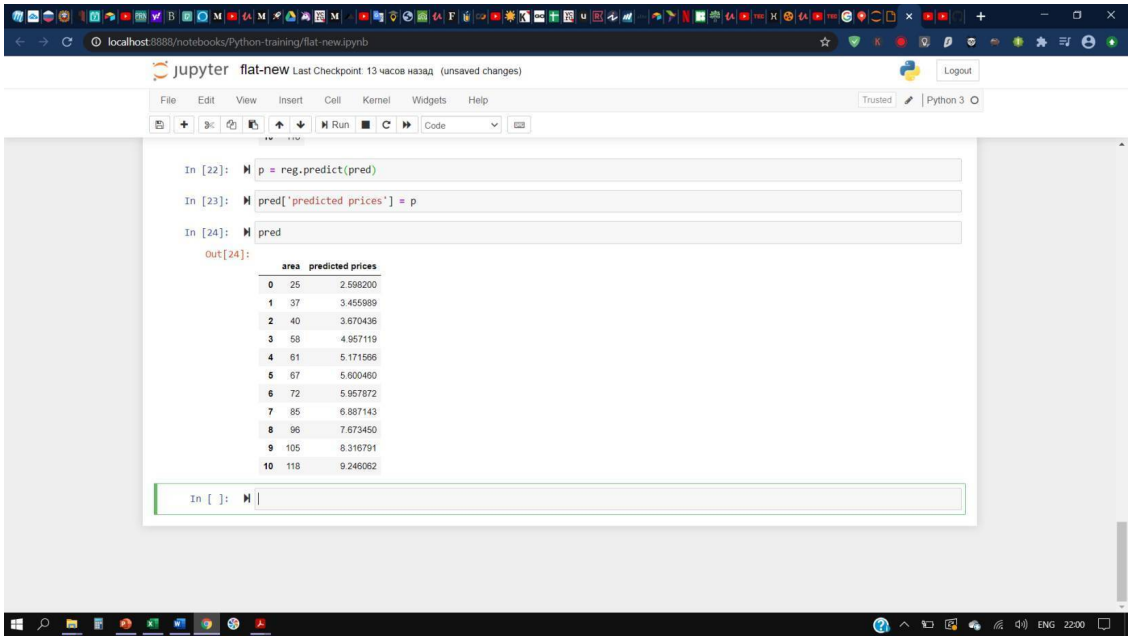
Для этого сначала давайте мы сохраним наши предсказанные цены в качестве переменной **p**.

p = reg.predict(pred)

А теперь создадим новую колонку в нашей таблице и вставим туда эти цены. Для этого мы просто пишем название нашей таблицы pred, потом в квадратных скобках и в кавычках название новой колонки predicted prices, то есть предсказанные цены и после этого мы назначаем в эту колонку те предсказанные цены, которые мы получили выше. Мы их обозначили буквой **p**.

pred['predicted prices'] = p

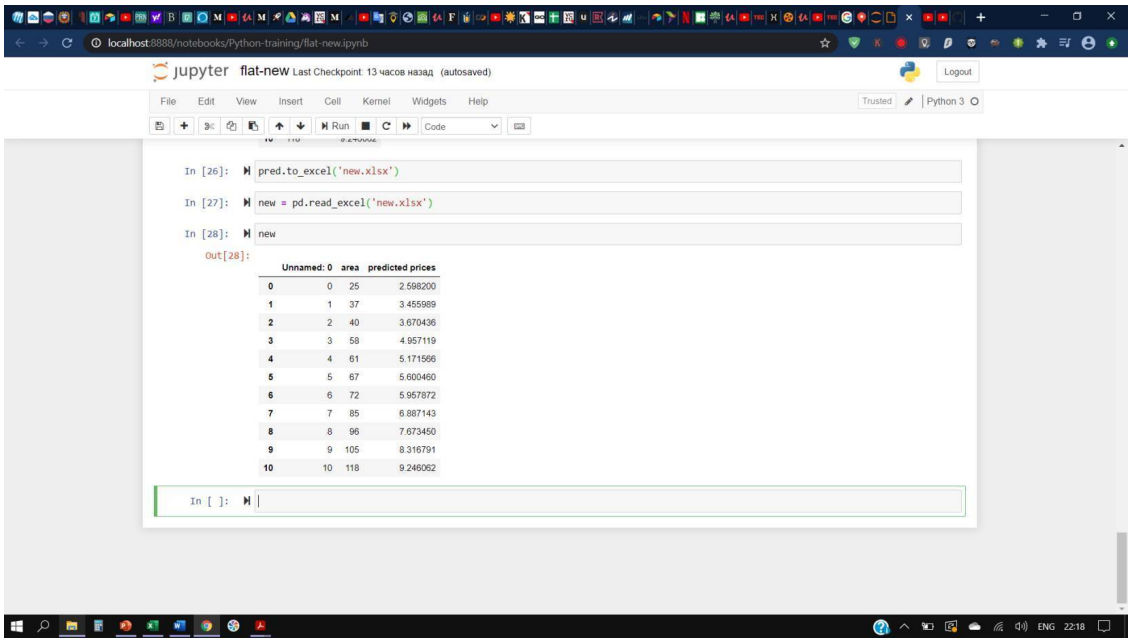
Давайте проверим, что получилось.



Теперь давайте просто сохраним этот файл в Excel с помощью следующей формулы:

pred.to_excel("new.xlsx")

Теперь этот файл должен появиться в нашей папке в Юпитере. Открываем его и проверяем. Отлично, единственное, был также экспортирован индекс, то есть первая колонка, которая в принципе не нужна.



Можем ее убрать в самом Excel, либо через код это убирается вот так:

pred.to_excel("new.xlsx", index = False)

Проверяем опять, да индекс исчез, отлично.

Итак, что мы прошли в этой главе.

Во-первых, при работе с проектами машинного обучения нам практически всегда придется импортировать библиотеки перед началом каждого проекта. Это библиотеки Pandas, Numpy, Matplotlib, Scikit-learn и другие. Они позволяют работать с табличными данными,

визуализировать их на графиках и использовать уже готовые шаблоны различных алгоритмов машинного обучения.

Мы также научились загружать файлы Excel в наш юпитеровский ноутбук и визуализировать данные из таблицы на графике.

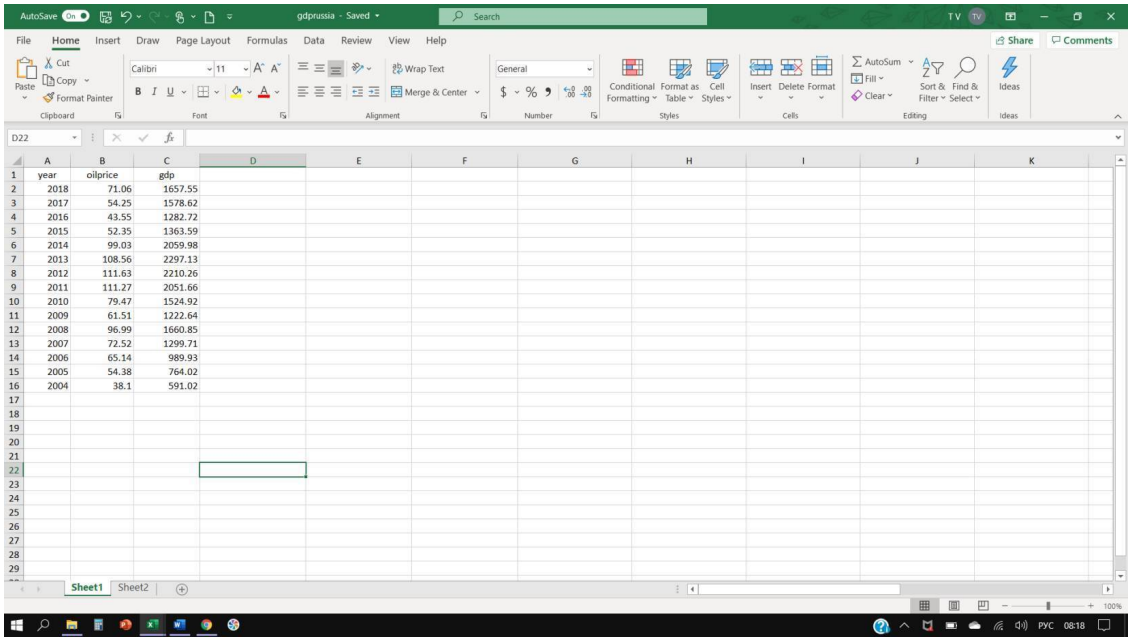
Мы научились тренировать нашу модель с помощью имеющихся данных и с помощью алгоритма линейной регрессии и предсказывать величины для новых данных. Ну и, кроме того, мы научились создавать новые колонки в табличках в Питоне и заносить туда наши предсказанные данные и потом сохранять это в файле Excel на своем компьютере.

Итак, теперь ваше самостоятельное задание.

Все мы знаем, что существенную долю бюджета России составляют доходы от продажи энергоресурсов, нефти и газа. Давайте в качестве самостоятельного задания спрогнозируем зависимость ВВП России от цен на нефть. Для этого мы будем использовать данные за прошедшие 15 лет.

Я подготовил для вас файл в Excel (gdprussia.xlsx), вы можете его скачать по ссылке с моего аккаунта на Github (https://github.com/timurkazantseff/machine-learning-101/). Если кому-то интересно, данные я брал с сайтов Всемирного банка, там очень много статистической информации по экономике и другим аспектам жизни в разных странах, а также с сайта Statista.com.

Файл выглядит следующим образом:



year	oilprice	gdp
2018	71.06	1657.55
2017	54.25	1578.62
2016	43.55	1282.72
2015	52.35	1363.59
2014	99.03	2059.98
2013	108.56	2297.13
2012	111.63	2210.26
2011	111.27	2051.66
2010	79.47	1524.92
2009	61.51	1222.64
2008	96.99	1660.85
2007	72.52	1299.71
2006	65.14	989.93
2005	54.38	764.02
2004	38.1	591.02

Получается, вам необходимо будет загрузить данный файл в ваш новый ноутбук в Юпитере, отобразить эти данные в виде графика, потом обучить модель и далее попробовать предсказать ВВП России в зависимости от разных цен на нефть.

На этом пока все. В следующей главе мы покажем пример выполнения этого задания. А пока попробуйте выполнить это задание самостоятельно. Успехов!

Предсказание ВВП в зависимости от цен на нефть

Итак, как ваши успехи? Удалось ли обучить модель для предсказания ВВП России в зависимости от цен на нефть? Давайте посмотрим, как будет выглядеть наш код.

Создаем новый файл, переименовываем его как нам надо. Сразу же первым делом загружаем библиотеки в наш проект.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
```

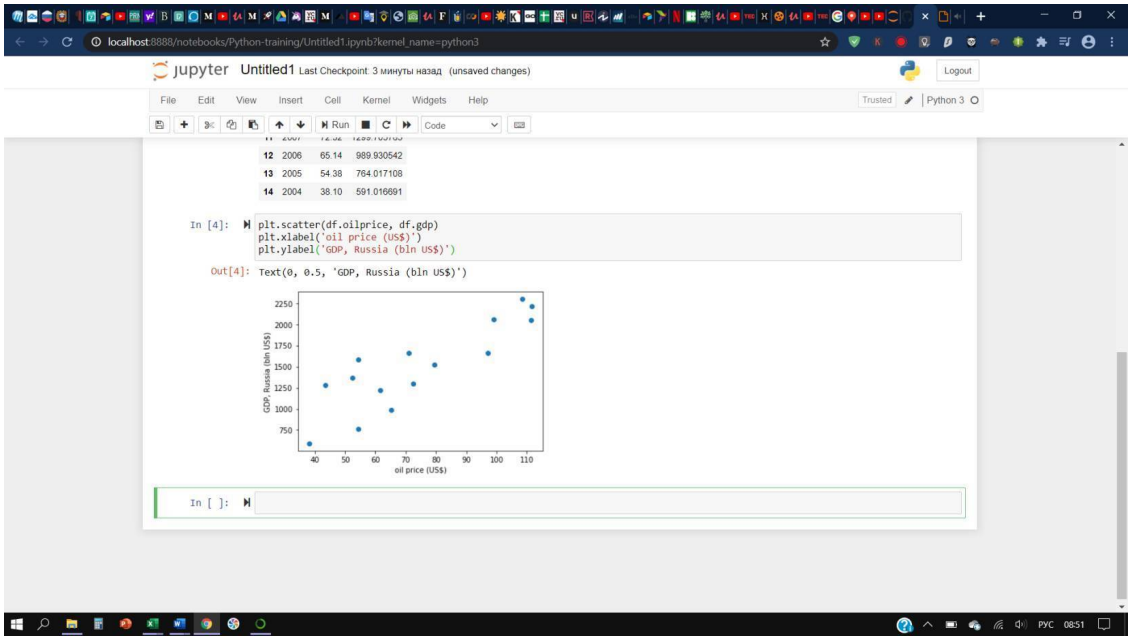
Дальше открываем в нашем ноутбуке файл Excel с помощью read.excel.

```
df = pd.read_excel('gdprussia.xlsx')
```

Начинаем строить график с помощью библиотеки **matplotlib**. Добавляем названия осей, стоимость нефти и ВВП России в миллиардах долларов.

```
plt.scatter(df.oilprice, df.gdp)
plt.xlabel('oil price (US$)')
plt.ylabel('GDP, Russia (bln US$)')
```

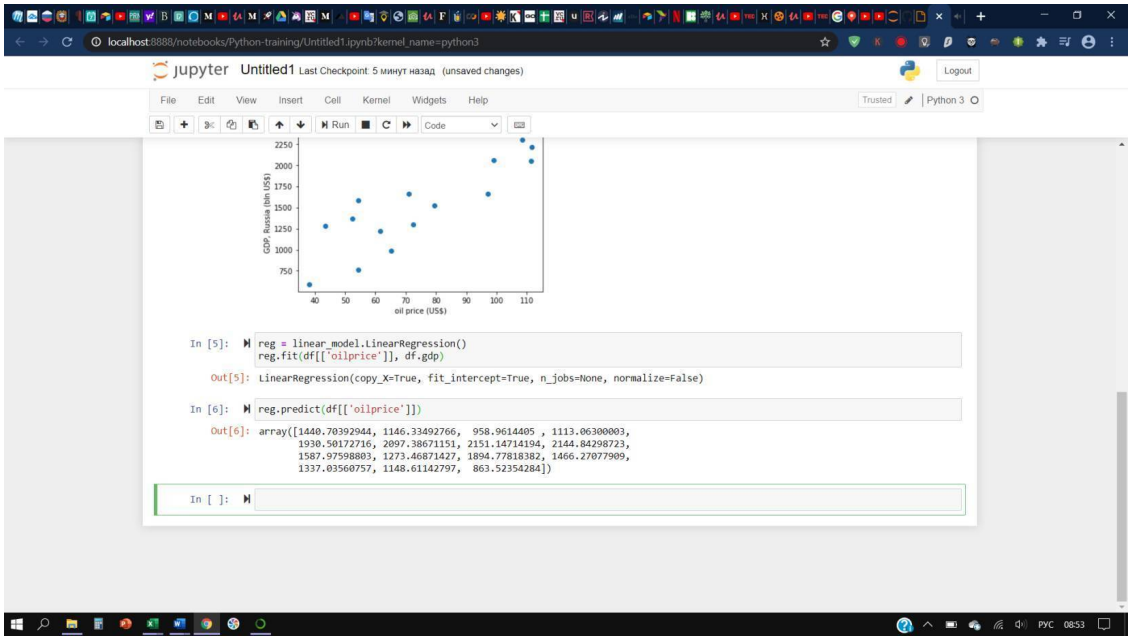
Так, выглядит достаточно правдоподобно. Как мы видим при одной и той же цене на нефть, в разные годы ВВП немного отличался. Возможно, это связано с тем, что Россия смогла диверсифицировать свою экономику либо по каким-то другим причинам.



Далее, запускаем нашу модель линейной регрессии. И с помощью команды **fit** заводим туда данные по цене на нефть, и ВВП страны.

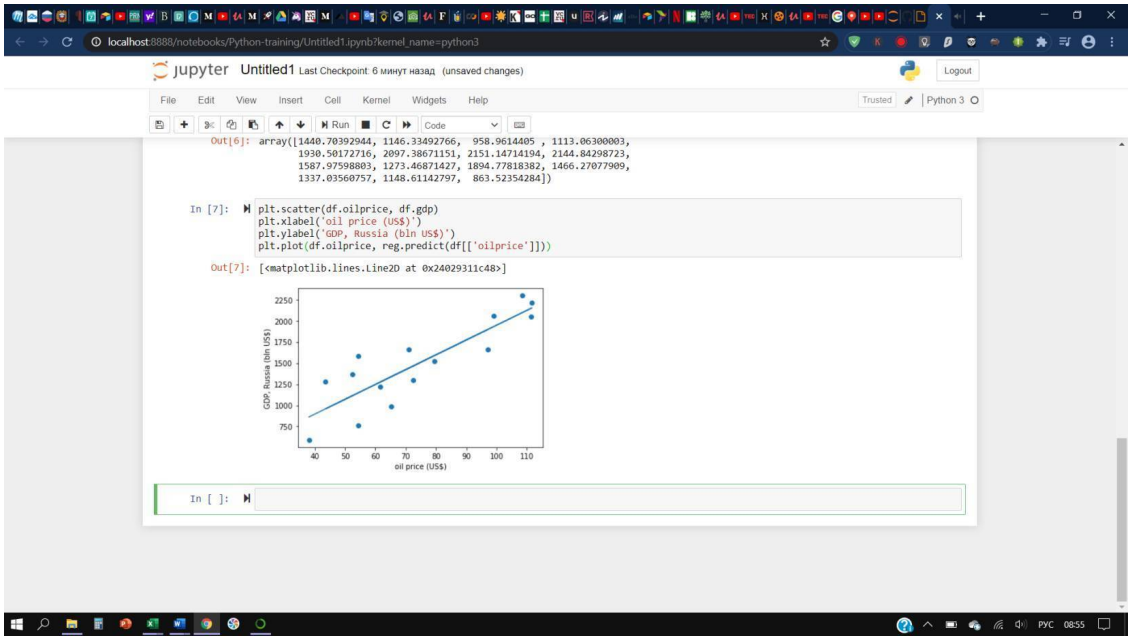
```
reg = linear_model.LinearRegression()
reg.fit(df[['oilprice']], df.gdp)
```

Вот нам выдало подтверждение, что модель обучена. Теперь мы можем предсказывать. Давайте сначала предскажем по всем годам и сравним насколько близки получились предсказанные значения с истинными.



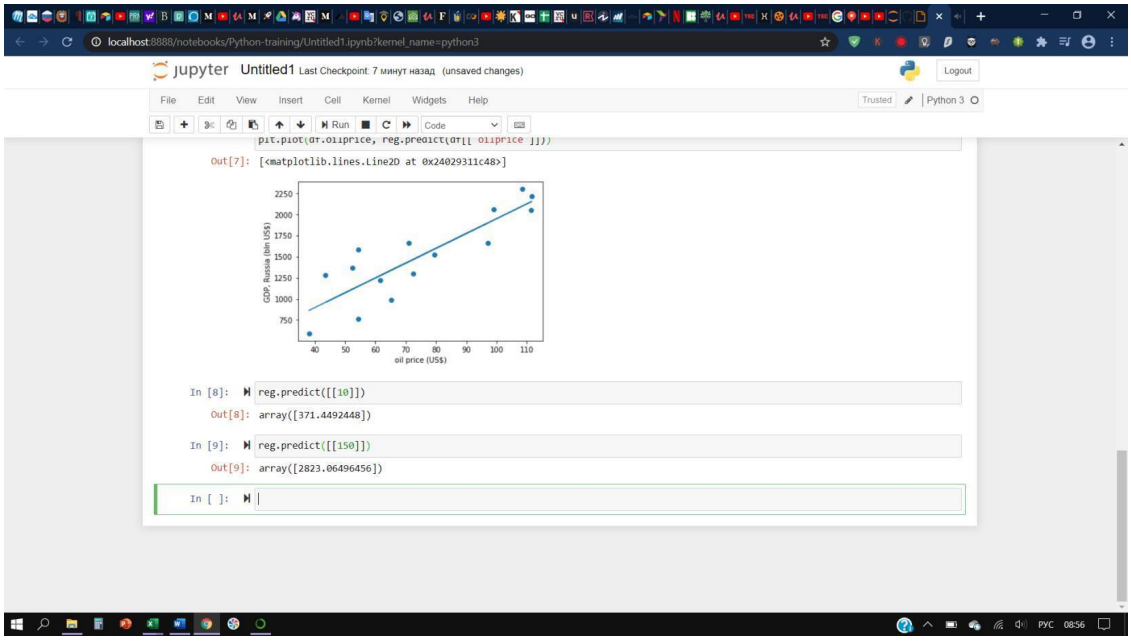
Ну разброс, конечно, немаленький, примерно от 200 до 400 миллиардов. О чем это говорит? Скорее всего о том, что корреляция ВВП и цен на нефть имеется, но далеко не 100%. Ну и это конечно очевидно, потому что российская экономика все-таки не полностью зависит от цен на энергоносители.

Можем построить линию, которая покажет как выглядит наша модель с помощью команды **plt.plot**.



Как видите разброс достаточно немаленький.

Давайте попредсказываем отдельные цены. Например, какой будет ВВП, если цена 10 долларов? 370 миллиардов, как-то очень мало для России. Ок, а если 150. Почти три триллиона, вот это уже нормально.



Кстати, сейчас мы делали предсказания основываясь только на цене на нефть. А давайте включим в нашу модель еще и зависимость от года.

Ведь давайте посмотрим еще раз на наши изначальные данные. Видите, в 2005 и 2017 году цена на нефть была одинаковая – около 54 долларов, однако ВВП кардинально отличается. В 2005 году всего 764 миллиарда, а в 2017 уже больше полтора триллиона. То есть за это время наша экономика все-таки стала сильнее и, возможно, менее зависима от цен на нефть.

Итак, как можно обучить нашу модель предсказывать по двум признакам. Наша модель в таком случае будет выглядеть следующим образом.

Y = aX + bZ + c,

где **Y** это наш ВВП, **X** – это цена на нефть, **Z** – это год, и **a**, **b**, **c** – это числовые коэффициенты.

А теперь как это указать в нашем коде? На самом деле очень просто. Помните, у нас было две строчки кода, которые мы как раз и использовали для обучения модели:

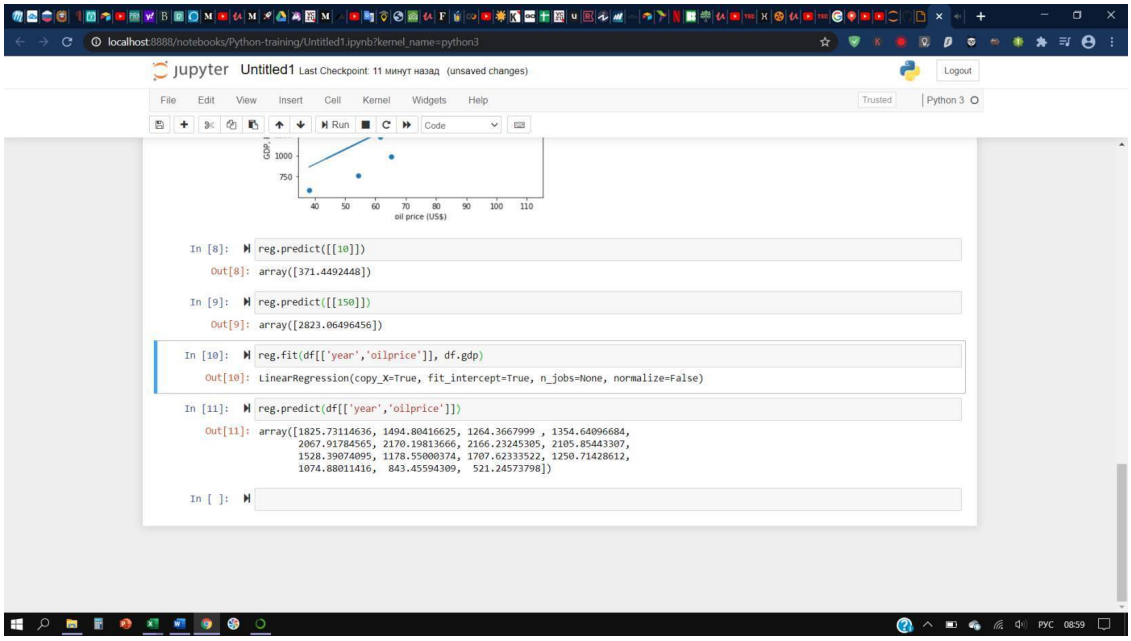
reg = linear_model.LinearRegression()

reg.fit (df[['oilprice']], df.gdp)

Мы просто в **reg.fit** добавляем еще один фактор через запятую, слово **year** в кавычках.

reg.fit(df[['year','oilprice']], df.gdp)

Нажимаем Enter – модель обучилась, а теперь давайте попредсказываем. Сначала весь ряд по всем годам.



Вот видите – уже намного лучше по точности. Разброс всего несколько десятков миллиардов. Давайте попробуем предсказать, какой будет ВВП, например, в 2025 году, если цена на нефть будет 100 долларов.

reg.predict([[2025,100]])

Почти 2 триллиона 700 миллиардов и почти на один триллион больше, чем сейчас.

Ок, я думаю, на этом пока все по линейной регрессии. Можете также попробовать найти статистические данные, например, по курсу валют или цене акций и попробовать создать и обучить модель на основе этих данных, чтобы делать предсказания в дальнейшем.

Выжившие на Титанике. Модель классификации с помощью Метода опорных векторов

В этой главе мы будем решать задачу классификации. Мы будем предсказывать какие из пассажиров выжили при крушении Титаника.

Для начала давайте вспомним что-такое задача классификации в машинном обучении. Как можно понять из названия, классификация используется для того, чтобы отнести тот или иной объект к определенному классу. Например, в нашем случае мы будем относить пассажира к категории либо выживших, либо нет. Задачи классификации могут использоваться, например, чтобы понять, относится ли входящее письмо к спаму или нет, либо чтобы понять, давать ли банку кредит потенциальному заемщику.

Если классов всего два, то задача называется бинарной классификацией. Если классов несколько – то многоклассовая классификация.

Если сравнить с регрессией, то в регрессии у нас нет классов, мы просто предсказываем числовую величину, например, в предыдущих эпизодах мы предсказывали стоимость квартиры или объем ВВП нашей страны, то есть конкретное число. А в классификации количество ответов или классов ограничено, мы сами предоставляем эти классы компьютеру, и компьютер определяет к какому из этих классов относится новый объект.

Итак, давайте приступим к решению нашей задачи классификации.

1.Сначала загружаем данные с файлом о выживших с Титаника с сайта Kaggle. Это очень полезный сайт, здесь вы можете найти очень много примеров для машинного обучения с объяснениями и вариантами решения другими пользователями.

www.kaggle.com/c/titanic/data

2. Далее, загружаем этот файл с помощью upload в нашу папку, где хранятся наши файлы для Юпитера.

3. Создаем питоновский ноутбук в Юпитере.

4. Импортируем библиотеки Pandas и Numpy.

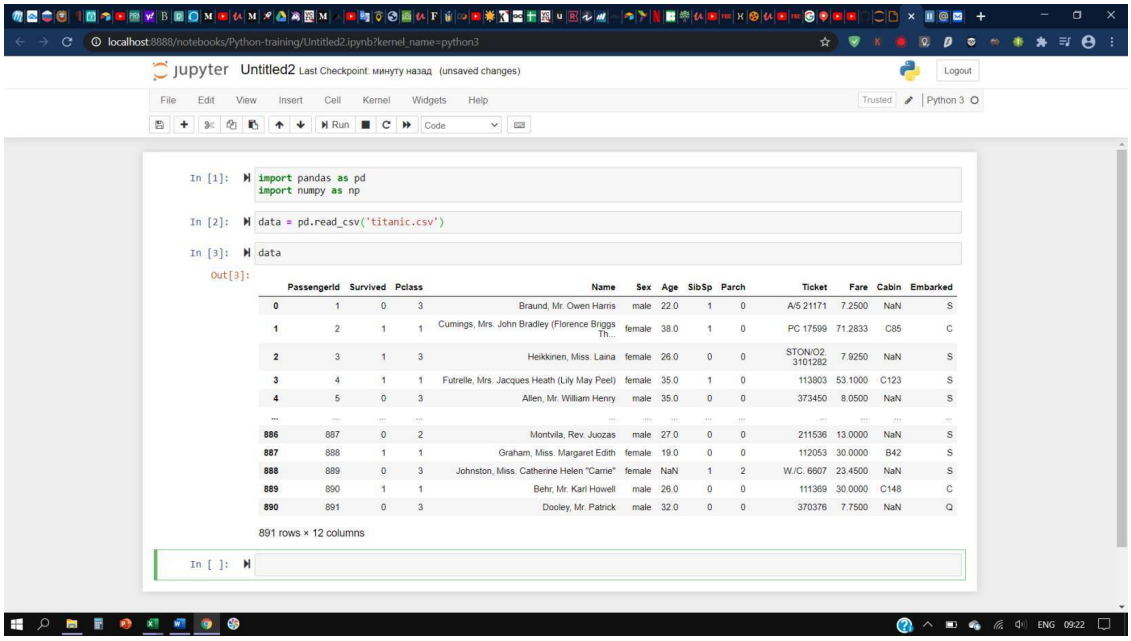
import pandas as pd

import numpy as np

5. Загружаем в ноутбук наш файл с Титаником.

data = pd.read_csv('titanic.csv')

Теперь давайте посмотрим, как выглядит наш файл.



Как видим, здесь данные по 891 пассажиру, есть данные по их полу, возрасту, какой класс каюты они занимали, и для нас самая важная колонка будет Survived. Как можно догадаться, 0 это если пассажир не выжил, и 1 – если он спасся.

Давайте назовем колонку Survived нашей целевой колонкой.

columns_target = ['Survived']

И возьмем еще четыре колонки, которые, по нашему мнению, влияли на то, смог ли спастись пассажир или нет. Эти колонки будут: пол пассажира, его возраст, класс каюты и тариф билета. Назовем их колонками для обучения.

columns_train = ['Pclass', 'Sex', 'Age', 'Fare']

Обратите внимание, названия колонок мы вписываем именно так, как они указаны в файле, то есть я имею в виду, там где заглавные буквы, то надо вводить именно заглавные буквы, в противном случае, данные не прочитаются.

Давайте теперь создадим переменные, которые будут сохранять данные, которые хранятся в этих колонках.

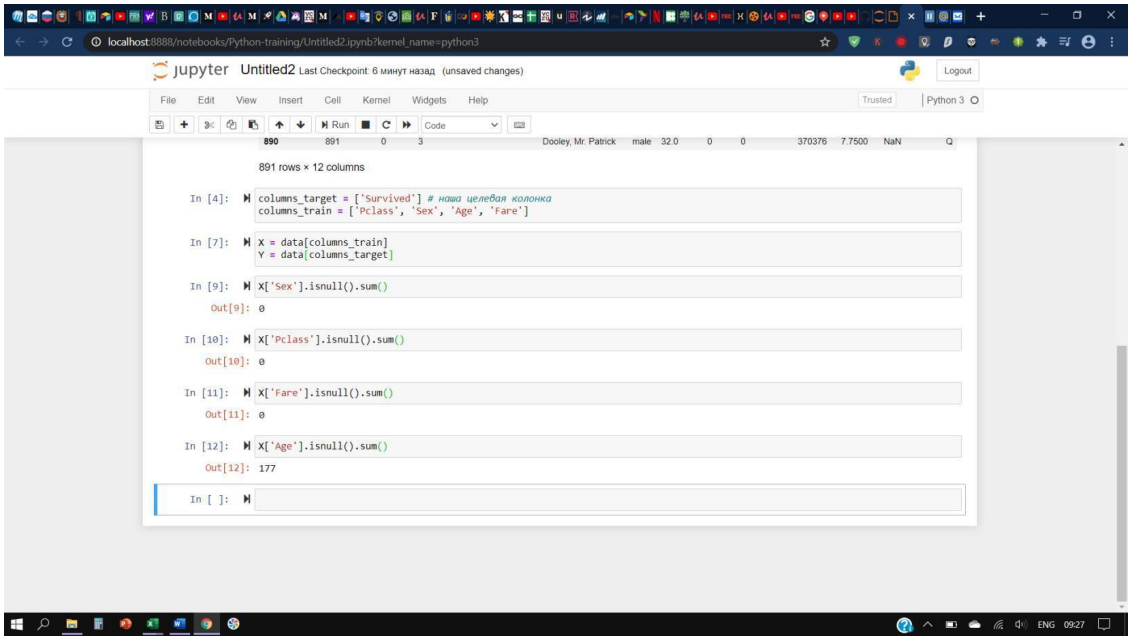
X = data[columns_train]

Y = data[columns_target]

Еще очень важный момент в Машинном обучении. Как вы понимаете, когда мы создаем модели, важны не только правильные алгоритмы, но и качество данных, на которых мы их обучаем. Имеется в виду, чтобы в наших данных не было пробелов или значений, которые там точно не могут быть. Например, при заполнении этой таблицы могли просто забыть вписать значение в какую-то ячейку, или вдруг значение возраста нечаянно перенесли в ячейку с тарифом. Поэтому в крупных проектах, когда дело идет на большие суммы, эти данные очень строго проверяют.

Что же мы можем сделать чтобы проверить качество данных в данном проекте. Давайте как минимум проверим, что у нас нет пустых ячеек. Это делается с помощью такой формулы:

X['Sex'].isnull().sum()



Вот нам выдало, что в колонке **Age** есть 177 пассажиров у которых пол не указан. Что делать?

Если мы удалим всех этих 177 пассажиров, это может сильно повлияет на нашу выборку, потому что все-таки это уже достаточно большое количество относительно всей выборки. Поэтому что мы можем сделать? Мы можем придать им среднее или медианное значение по данной колонке.

В **pandas** для этого есть специальная функция, которая заполняет эти пустые ячейки с помощью медианных или средних значений по колонке. Давайте воспользуемся средними значениями.

X['Age'] = X['Age'].fillna(X['Age'].mean())

Нам выдало небольшое розоватое предупреждение, но его можно смело игнорировать. Давайте лучше проверим еще раз заполнены ли теперь все ячейки в колонке Age.

X['Age'].isnull().sum()

Да, теперь пустых ячеек в этой колонке нет, можем ли мы теперь двигаться дальше? Не совсем. Надо провернуть еще одну манипуляцию.

В нашей модели в **sklearn** мы не можем использовать переменную типа “строка” как категориальную переменную. Что это значит. То, что значения в колонке “пол пассажира” указаны как слово мужчина **male** или женщина **female**. Наша модель не сможет их использовать для обучения. Нужны только числовые данные. Поэтому что мы сделаем. Мы просто переименуем эти названия в нули и единицы. Давайте заменим женщин на единицу, а мужчин на ноль.

Для этого нам потребуется создание словаря. Помните, в одной из предыдущих глав мы изучали как создавать словари? Словарь, это когда вы говорите, что окей, вот эта переменная обозначает или равна вот этой другой переменной. Как в обыкновенных словарях по переводу с одного языка на другой. В словарях используются фигурные скобки и там две переменные объединены двоеточием. Назовем наш словарь **dict**.

dict={'male':0, 'female':1}

X['Sex'] = X['Sex'].apply(lambda x:dict[x])

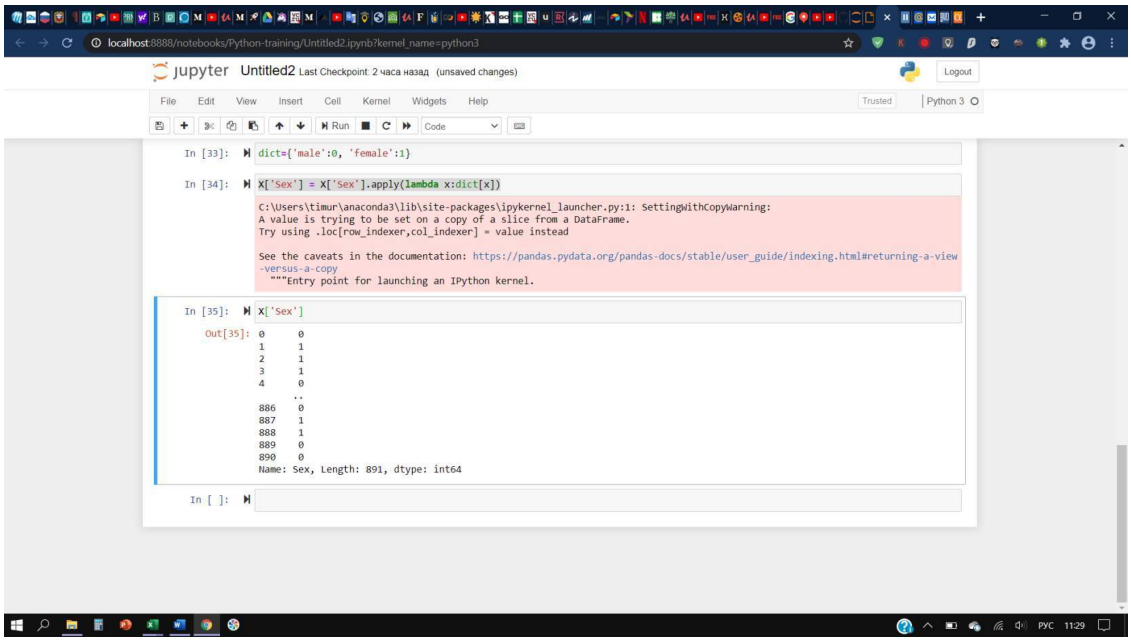
Вот это слово **lambda** как раз и обозначает, что мы как бы зеркально заменяем наши значения, как это указано в нашем словаре.

Кстати, если у вас встречаются ошибки, перед тем как пытаться найти в интернете возможные причины, просто попробуйте заново прогнать код с самого начала,

либо посмотрите внимательно, везде ли вы правильно поставили разные виды скобок, заглавные буквы и т.д.

Давайте проверим как теперь выглядит наша колонка с полом пассажиров:

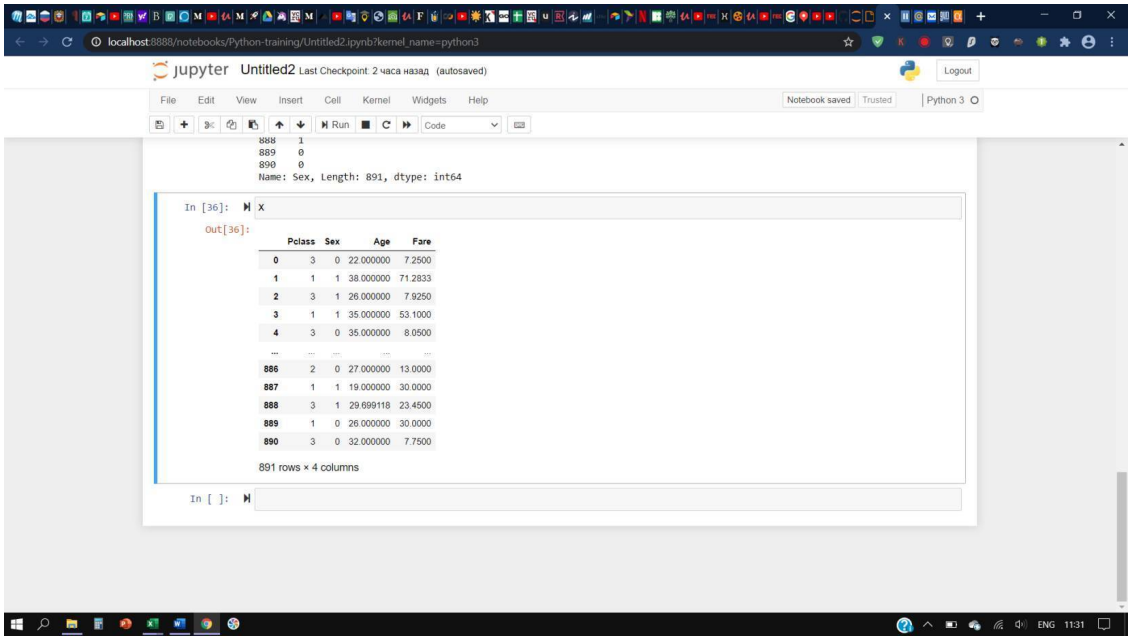
X['Sex']



Отлично, теперь везде нули и единицы, то что и надо.

Давайте еще раз проверим наши данные, как помните, для тренировочного датасэта мы оставили всего 4 колонки, возраст, пол, класс кают и тариф билета.

X



Итак, что еще важно в машинном обучении, это разделять наш датасэт на две части, первую – это тренировочная часть, но основе которой мы будем обучать нашу модель. И вторая

часть – это тестовая, по которой мы будем проверять как хорошо справляется наша модель, ее точность.

Изначальная выборка (датасэт)

Давайте разобьем нашу выборку на четыре части X_train, Y_train, и X_test и Y_test.

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=42)

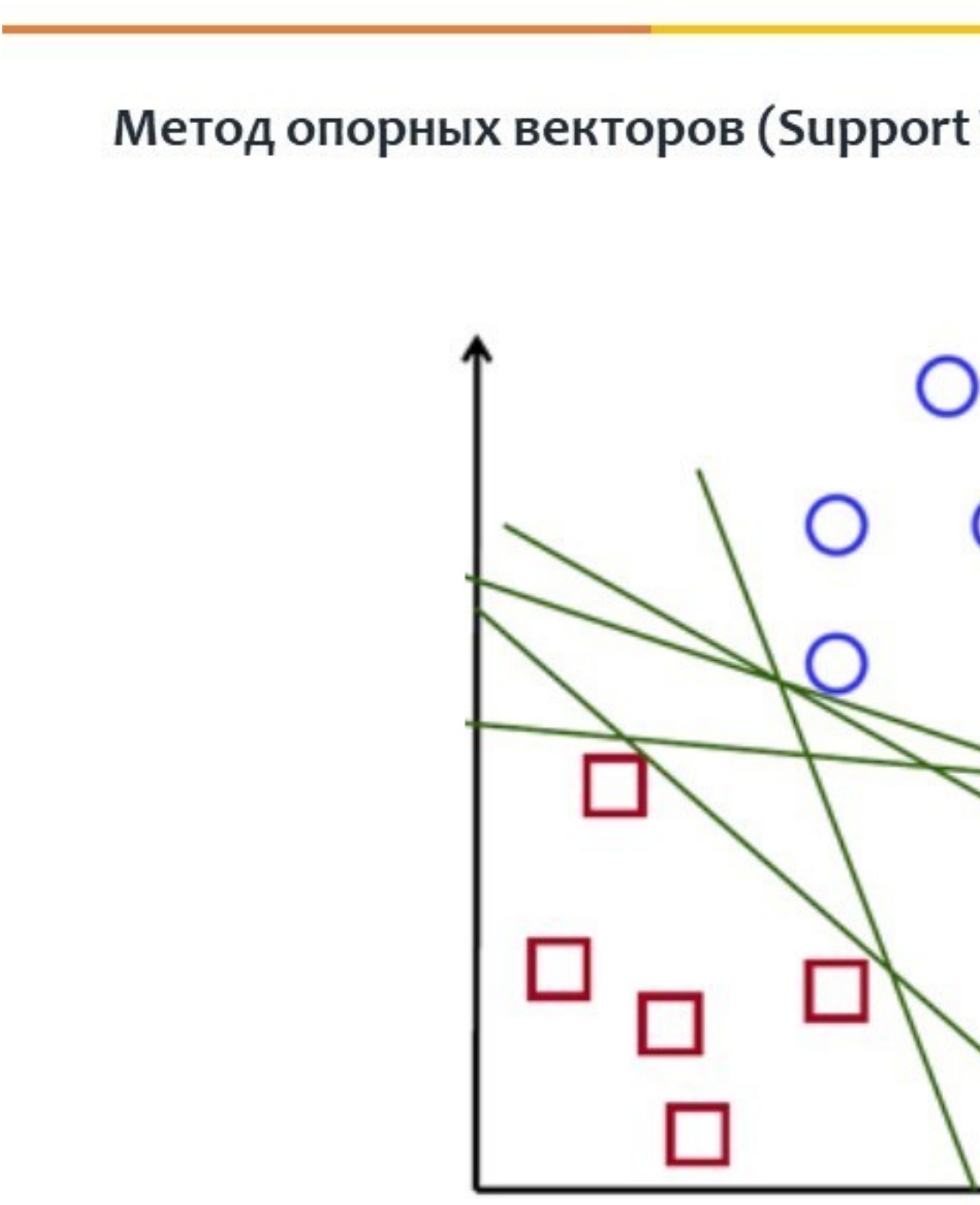
Это стандартные показатели: мы берем размер тестовой выборки как одну треть, а показатель random state в качестве 42, можете рассматривать их как базовые показатели по умолчанию.

Так, теперь нам надо обучить нашу модель на нашей обучающей выборке.

Наша модель будет основана на методе опорных векторов, или Linear Support Vector Machine, который линейно разделяет наши данные на разные группы.

Данный метод заключается в том, что мы пытаемся найти такую линию (или в случае если у нас несколько классов такую гиперплоскость), расстояние от которой до каждого класса максимально. Такая прямая, или гиперплоскость, называется оптимальной разделяющей гиперплоскостью. Те же точки или как можно сказать представители классов или вектора, которые лежат ближе всех к разделяющей гипер-

плоскости, называются *опорными векторами*. Отсюда и происходит название данного



метода.

Чем хорошо программирование в Питоне, так это тем, что в нем есть много библиотек и готовых модулей, которые уже содержат все эти методы, то есть нам не надо самим составлять и высчитывать все эти формулы. Мы просто импортируем нужную нам модель в наш проект.

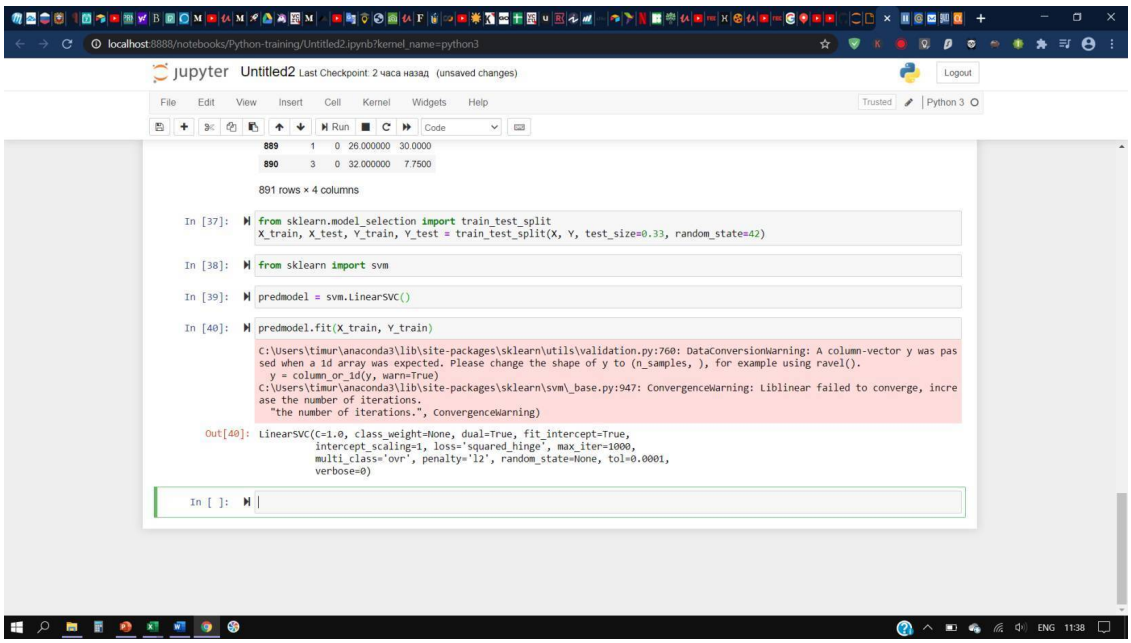
Поэтому давайте импортируем шаблон метода опорных векторов в наш ноутбук и назовем нашу модель predmodel.

from sklearn import svm

predmodel = svm.LinearSVC()

Теперь давайте обучим нашу модель с помощью нашей обучающей выборки и функции **fit**.

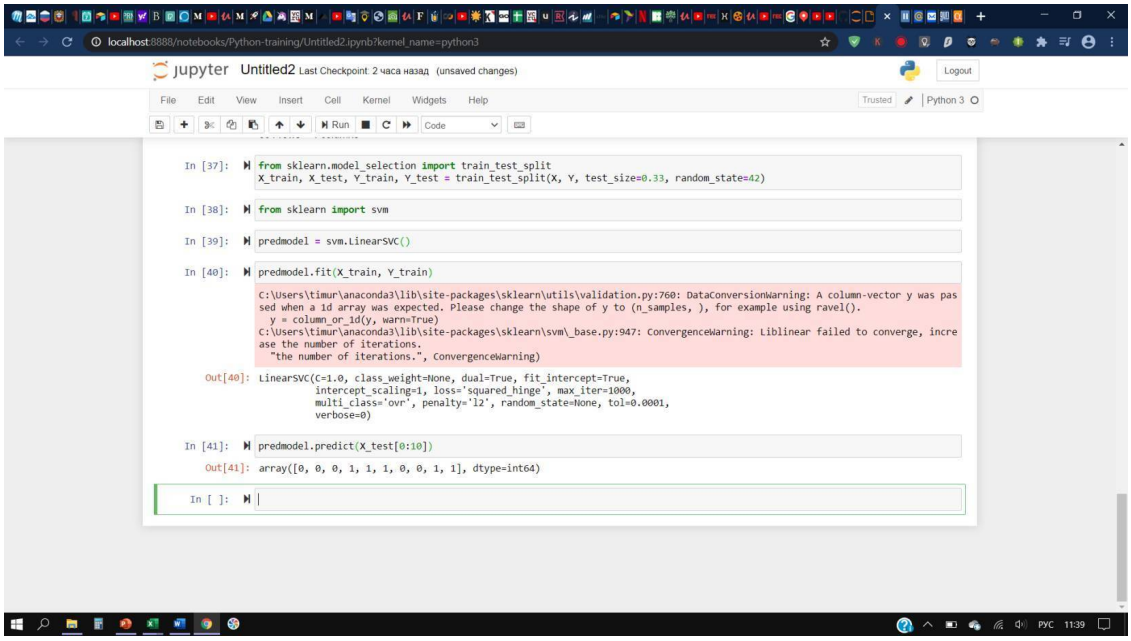
predmodel.fit(X_train,Y_train)



Отлично, нам выдало подтверждение, что модель обучена, опять же игнорируем эти розовые уведомления. Теперь самое время делать предсказания на тестовой выборке.

predmodel.predict(X_test[0:10])

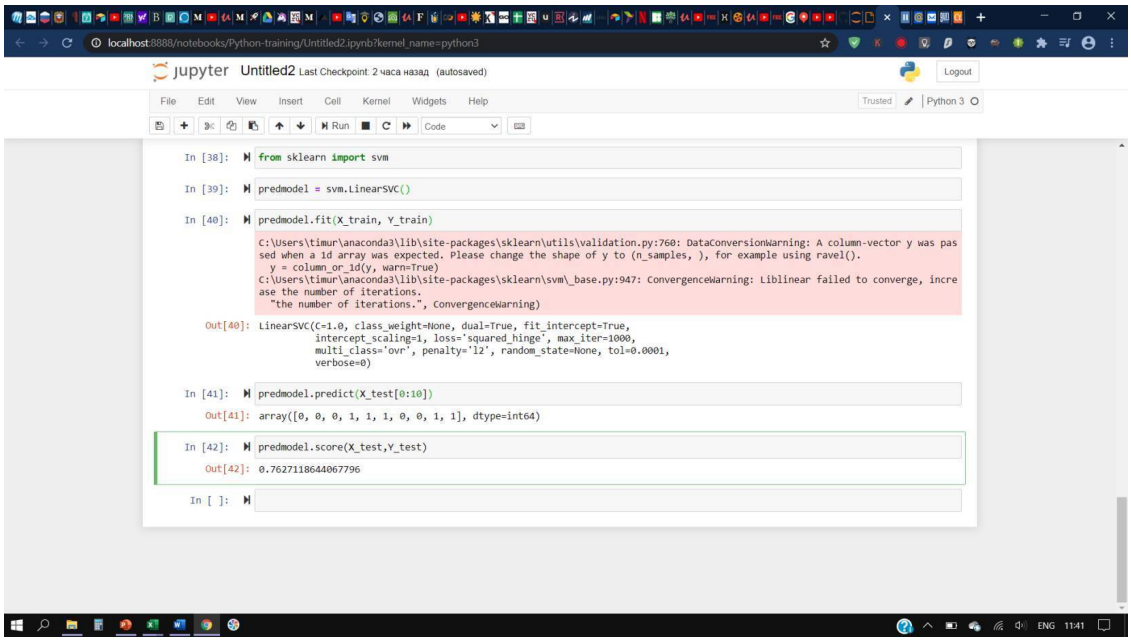
То, что в скобках указано 0 двоеточие 10 означает, что мы предсказываем первые 10 значений из нашей тестовой выборки.



Отлично, мы видим нули и единицы, что означает, выжил ли пассажир или нет, но как мы узнаем, правильно ли модель предсказала эти значения.

Для этого используем функцию **score**.

predmodel.score(X_test,Y_test)



Нам показало почти 76,3%, что достаточно неплохо для нашей первой модели.

Кстати, вы можете прогнать этот код заново, и вам выдаст совсем другую точность. Потому что когда машина разделяет вашу выборку на обучающую и тестовую, она делает это рандомно, то есть случайным образом, поэтому каждый раз точность будет разной.

Также, помните там, где мы нашли 177 пустых ячеек с возрастом, мы вставили в них средний возраст, попробуйте, например, вставить медианный возраст с помощью команды **median** и посмотрите какой будет результат.

X['Age'] = X['Age'].fillna(X['Age'].median())

На этом пока все, Питоновский код к этой задаче находится на моем аккаунте в Github (https://github.com/timurkazantseff/machine-learning-101), но в любом случае, попробуйте самостоятельно напечатать данный код, чтобы закрепить пройденный материал.

Решение задачи Выживших на Титанике с помощью модели Дерева решений, Случайного леса и Бэггинга

В прошлой главе мы решали задачу выживших на Титанике с помощью метода опорных векторов. Согласно этому методу, мы находили такую оптимальную линию, расстояние от которой до каждого класса было бы максимальным.

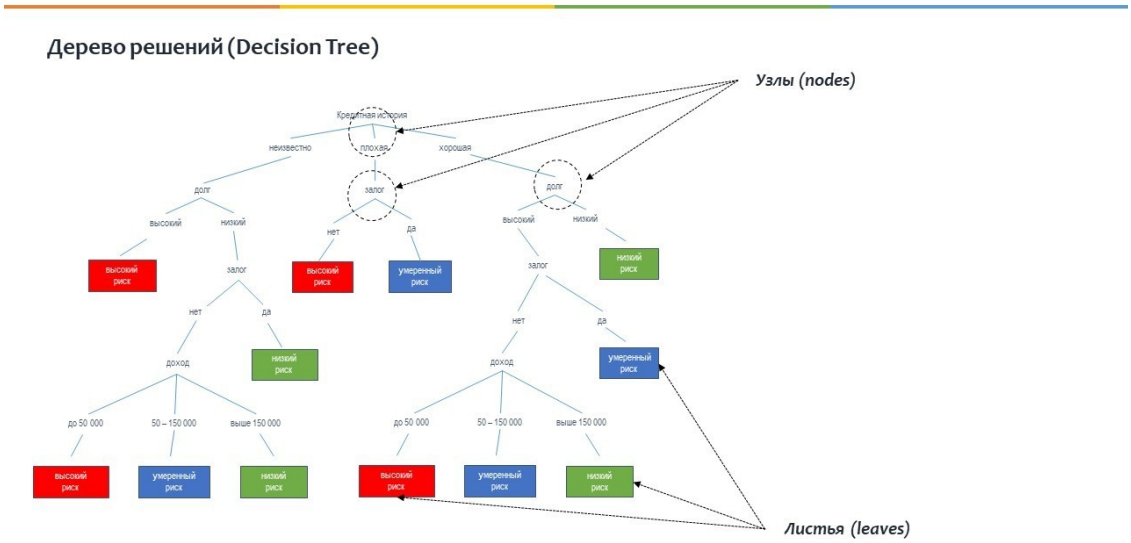
Задачи классификации также часто решают с помощью другого метода, который называется дерево решений. В теоретической части этой книги мы уже упоминали о дереве решений.

Дерево решений выглядит как алгоритм или инструкция, что делать в какой ситуации. Выявляются параметры, которые играют существенную роль, и в зависимости от этих параметров строится модель, которая в итоге выдает ответы или распределяет наши объекты в тот или иной класс.

Самый типичный пример – это кредитный скоринг, когда банк пытается решить выдать или нет кредит потенциальному заемщику. И на основе того, какой у него возраст, какой у него ежемесячный доход и расходы, имеются ли на данный момент другие кредиты, женат он или холост, и в зависимости от других дополнительных параметров, в итоге нам выдается ответ, давать кредит или нет. Это задача бинарной классификации, когда класса всего два. В принципе, как и в нашем примере с выжившими с Титаника, тоже класса на выходе всего два.

Когда компьютер строит модель дерева решений, он пытается понять какие из факторов являются самыми важными в первую очередь, потом какие важны в следующую очередь и так далее, вплоть до того, когда уже становится понятным к какому классу можно отнести данный объект.

Последние ячейки, которые уже обозначают класс, к которому принадлежит объект, называются листьями дерева решений. Те места, где происходит разветвление, это узлы дерева. Ну и сама вся структура называется дерево решений, потому что действительно напоминает дерево с различными ветвями.



Давайте посмотрим еще на один пример. Есть такая игра, называется «20 вопросов». В ней на лоб человека клеят бумажку с названием известного человека, и надо угадать этого

человека за меньше чем за 20 вопросов. Вопросы должны быть такими, чтобы ответы на них могли быть только да/нет. Если вы будете сразу называть имена известных людей, например, скажете Билл Гейтс или Мадонна, то шанс достаточно небольшой, что вы угадаете, потому что таких известных людей миллионы.

Поэтому правильной стратегией будет, если вы начнете постепенно сужать вашу выборку по максимуму. Например, первым вопросом мог бы быть – это мужчина или женщина. Потом, ему больше 40 лет или меньше. Дальше, этот человек из мира спорта или нет? И так далее.

Итак, теперь зная, что такое деревья решений, давайте посмотрим, как мы можем их использовать, чтобы предугадать выжили или не выжили те или иные пассажиры на Титанике.

В нашем примере с Титаником мы даем компьютеру все данные по выжившим и невыжившим, и он также строит модель дерева решений. Мы имеем несколько признаков: пол и возраст пассажира, класс каюты, стоимость билета, и так далее. Он понимает, что главным узлом в дереве решений должен быть пол пассажира, потому что если это мужской пол, то с вероятностью 78% он не выжил. Далее, если это женщина, то для нашей модели становится важным в каком классе каюты она проживала. 3 класс – самый дешевый. Если не третий, а выше, то скорее всего выжила. И так далее, компьютер строит такую модель.

Как уже говорил, нам не надо знать точных математических вычислений для того, чтобы знать, как строится дерево решений. В библиотеке scikit-learn уже установлены все необходимые модули и наша задача будет сводиться к тому, чтобы правильно подать все необходимые данные, подчистить их, если необходимо, и просто ввести пару строчек кода для инициализации самого алгоритма.

Итак, давайте вернемся к нашему ноутбуку в Юпитере. У нас был файл titanic-prediction, где мы в прошлой главе использовали метод опорных векторов чтобы делать предсказания. Давайте продублируем его и переименуем в titanic-decision-tree, потому что большую часть кода мы менять не будем, нам потребуется только изменить пару строчек.

Итак, давайте откроем этот файл.

В самом начале, где мы импортируем библиотеки и модули, давайте добавим еще пару модулей для алгоритма дерева решений из Scikit learn.

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

Отлично, далее просто прокручиваем весь код, ничего не меняя.

Иногда при выполнении того или иного кода вам могут выходить розовые предупреждения. Я уже говорил, что их можно смело игнорировать. Но если вы не хотите, чтобы они постоянно появлялись, вы их также можете спокойно отключить с помощью вот этой строки кода:
pd.options.mode.chained_assignment = None

Итак, идем дальше.

После строчки, где мы поделили нашу выборку на обучающую и тестовую, мы можем вставить пустые строчки для того, чтобы обучить нашу модель с помощью алгоритма дерева решений.

Для этого, назовем нашу модель **clf** от слова **classifier**.

clf = tree.DecisionTreeClassifier(max_depth=5, random_state=21)

Max_depth означает глубину дерева, вы можете поэкспериментировать и поставить большую глубину, но так как у нас признаков не так много и выборка небольшая, думаю 5 будет достаточно. Чем больше глубина, тем точность предполагается будет больше, но в то же время это будет занимать больше времени, чтобы обучить модель. Поэтому в нашем примере ограничимся 5. А **random_state** вы можете указать любой, он нужен для того, чтобы потом вы могли заново воспроизвести именно такой результат, если захотите, поэтому можете указать его, либо тоже ничего не указывать, в нашем случае, это не так принципиально.

Итак, теперь обучаем нашу модель с помощью нашей тренировочной выборки.

clf.fit(X_train, Y_train)

Все нам выдало подтверждение, что модель обучена.

Давайте теперь проверим какую точность она выдает.

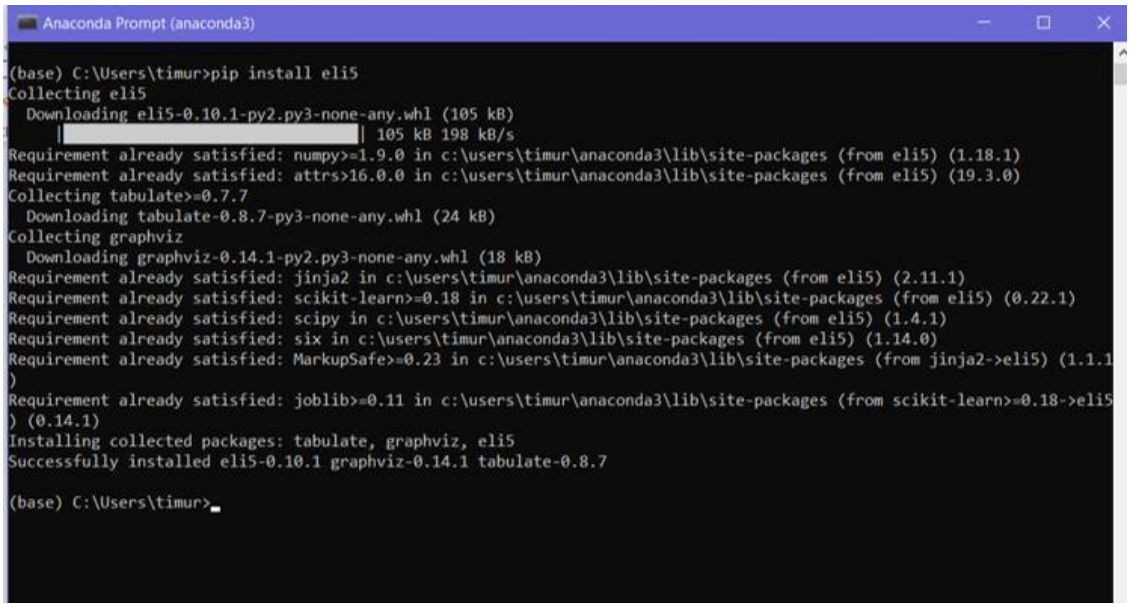
clf.score (X_train, Y_train)

Уже 81,7%, и это немного выше, чем было в предыдущий раз с помощью метода опорных векторов.

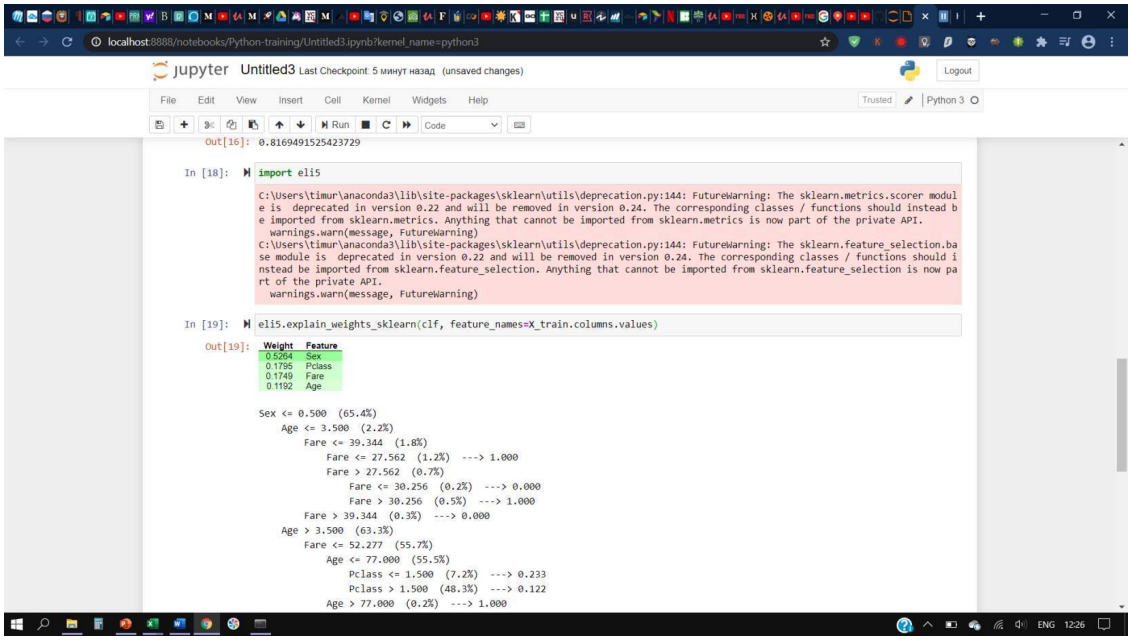
Что еще можно сделать. Вы можете посмотреть какие из признаков являются самыми главными по очередности в том, чтобы определить кто выжил согласно нашей модели, а кто нет. То есть мы можем определить вес каждого признака в нашей модели. У нас с вами было четыре признака: это возраст, пол пассажира, класс каюты и тариф билета. Чтобы понять, какой признак, согласно нашей модели, является определяющим, давайте сделаем следующее.

Сначала нам надо подключить библиотеку eli5. Для этого на новой строке напишите **import eli5**.

Если вдруг по каким-то причинам у вас выводит ошибку при вводе этой команды, попробуйте запустить Anaconda Prompt и в открывшемся окне написать **pip install eli5.**



И следующей строкой выводим веса каждого из признаков по нашей модели:
eli5.explain_weights_sklearn(clf, feature_names=X_train.columns.values)

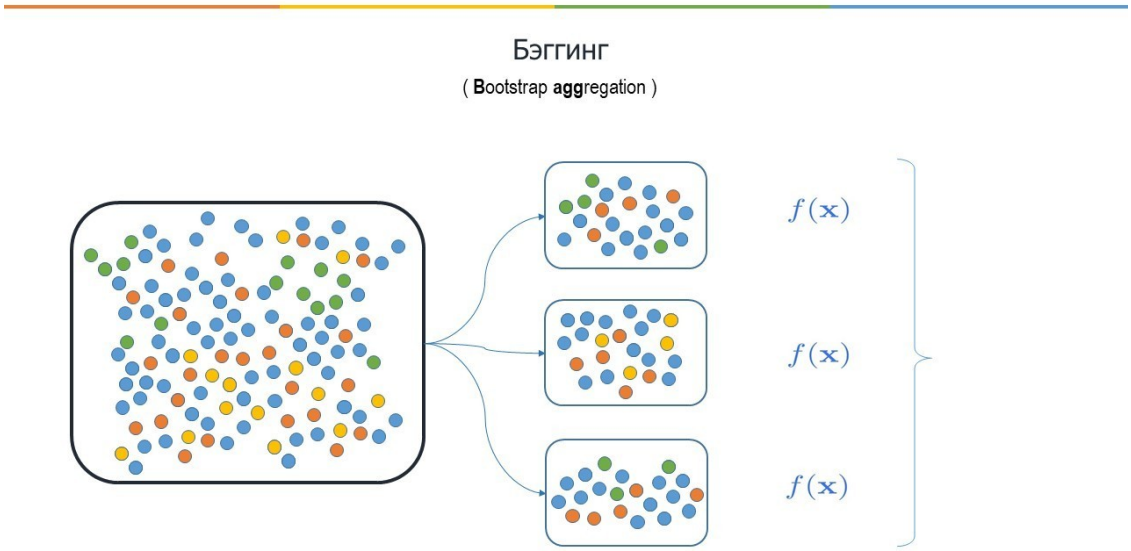


Итак, мы с вами видим, что в нашей модели самый большой вес имеет пол пассажира, далее следует класс каюты, потом тариф билета, и потом возраст.

И внизу в принципе представлено, как выглядит наше дерево решений и какому алгоритму следует компьютер, когда классифицирует выжил ли пассажир или нет.

Отлично, идем дальше. Теперь давайте попробуем другие алгоритмы для решения этой же задачи. Мы воспользуемся случайным лесом и бэггингом.

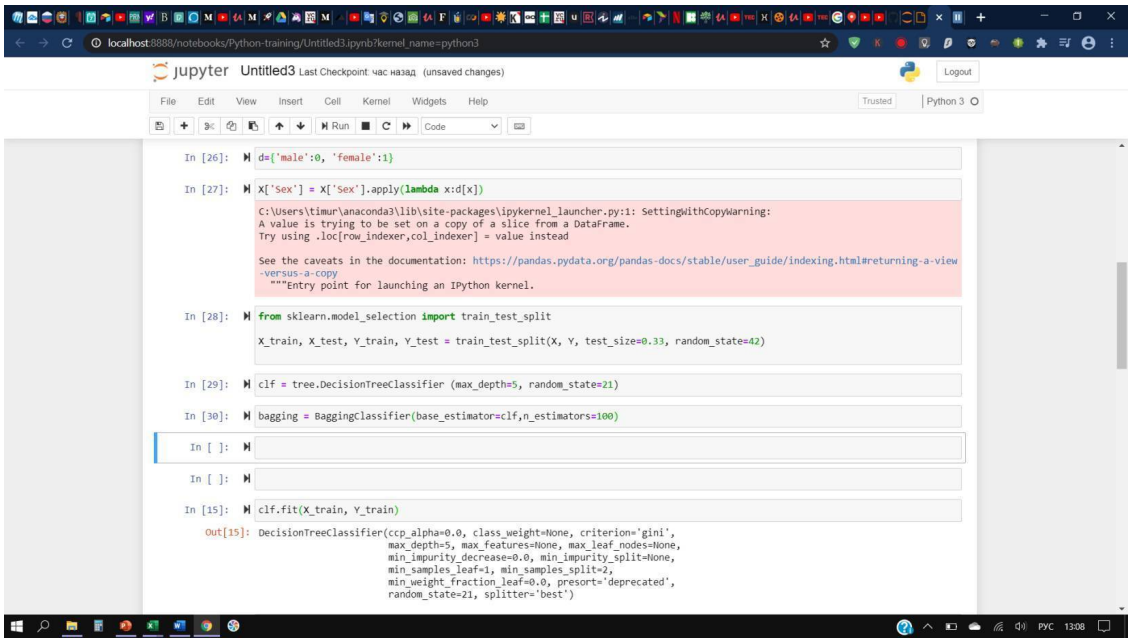
Давайте начнем с **бэггинга**. В бэггинге мы разбиваем наш основной датасэт на более мелкие датасэты. И в каждом мелком датасэте мы используем все те же признаки, как и в основном датасэте. И в итоге мы аггрегируем результаты.



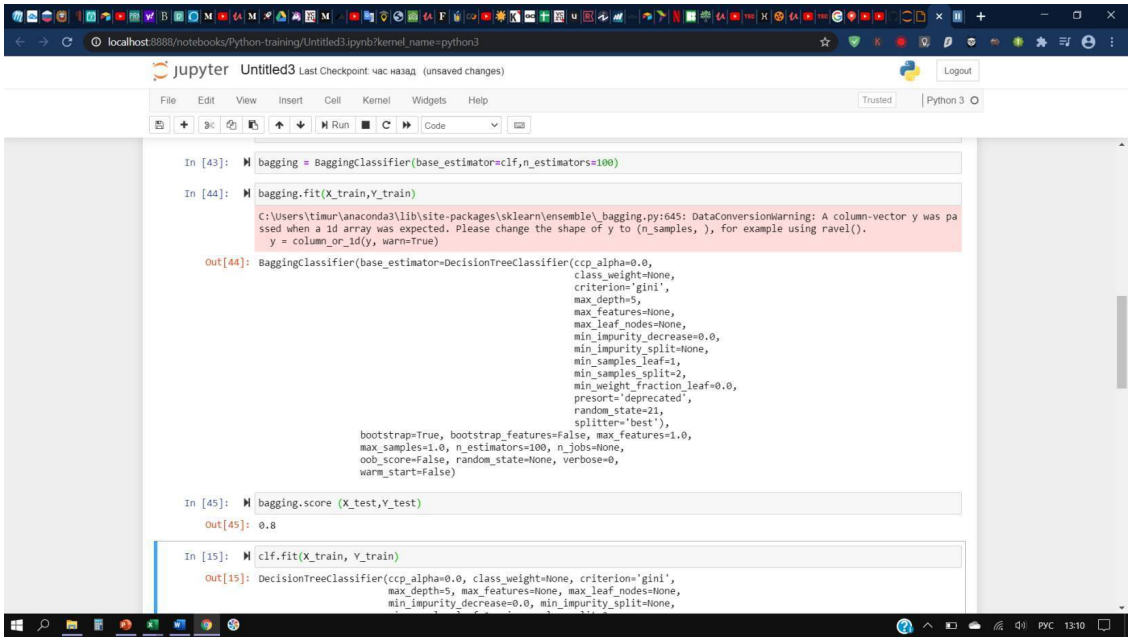
Итак, вернемся к нашему ноутбуку с моделью дерева решений. Мы будем использовать ее как основу для нашей модели бэггинга. Итак, для начала давайте сначала импортируем модуль бэггинга.

from sklearn.ensemble import BaggingClassifier

Прокручиваем код до того места, где у нас уже была модель, основанная на алгоритме дерева решений. Оставляем это как есть, но меняем наше обучение. Сначала задаем инструкцию для создания бэггинга и в скобках указываем, что за основную модель берем нашу модель из дерева решений. И потом обучаем уже бэггинг, а не нашу изначальную модель.



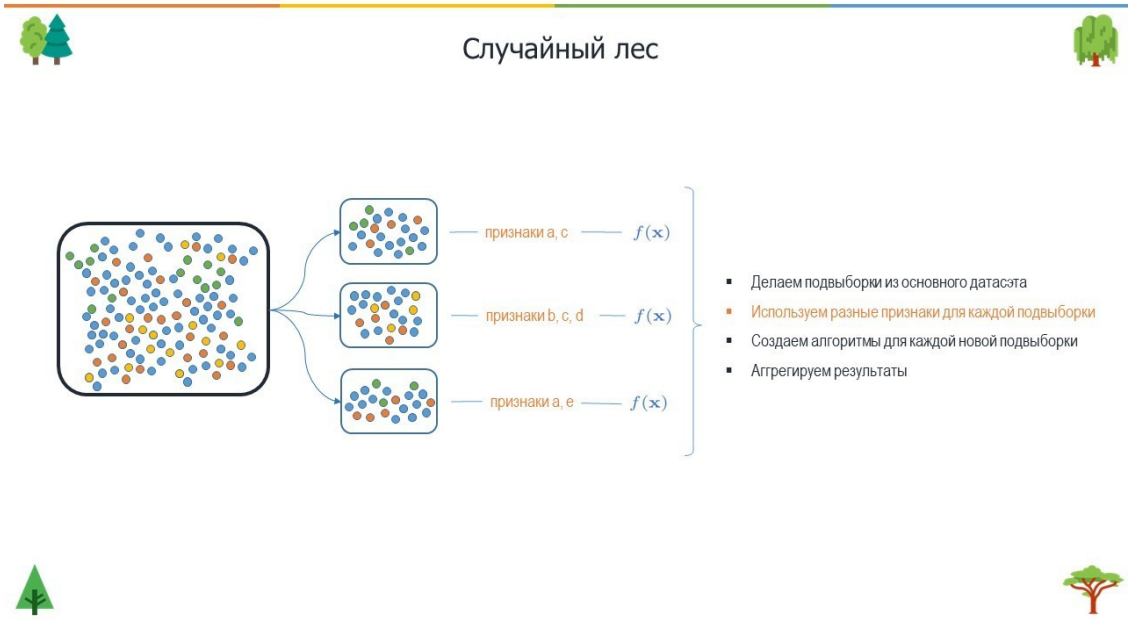
Теперь предсказываем и измеряем точность тоже указывая в строке именно бэггинг, а не саму изначальную модель.



В данном случае модель бэггинга показала точность (80%) выше модели опорных векторов, но немного ниже модели дерева решений.

Ок, теперь давайте попробуем **алгоритм случайного леса**.

Случайный лес – очень похож на бэггинг, когда из нашей основной выборки мы делаем несколько меньших по размеру подвыборок. Но в отличие от бэггинга, в каждой из этих подвыборок мы будем опираться не на все четыре признака, которые у нас имеются, а случайным образом будут выбираться только отдельные признаки. Например, у нас будет подвыборка с признаками пол и возраст, другая подвыборка с признаками возраст и класс каюты и т.д. Далее, результаты по этим подвыборкам будут агрегироваться и, предполагается, что мы сможем получить еще большую точность предсказаний. Итак, давайте посмотрим, как это будет выглядеть.



Для начала давайте продублируем наш файл titanic prediction и переименуем его. Напомню, это наш файл, где мы использовали модель опорных векторов.

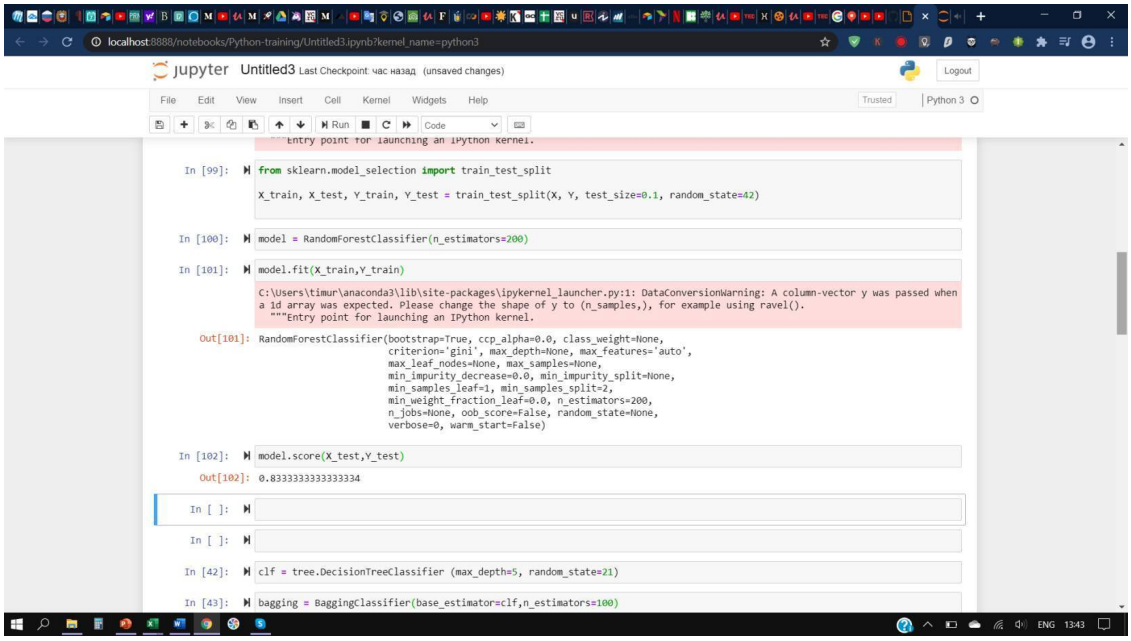
Для начала давайте импортируем модуль для случайного леса.

from sklearn.ensemble import RandomForestClassifier

Давайте опять прокрутим весь код и после того места, где мы разбиваем наш датасэт на обучающую и тестовую подвыборку, можем вставить несколько новых пустых строчек, чтобы было удобнее. Давайте обучим нашу модель с помощью случайного леса. Назовем нашу модель *model* и дадим ей шаблон случайного леса.

model = RandomForestClassifier(n_estimators=200)

Потом тренируем ее с помощью команды **fit** и нашей обучающей выборки **X_train** и **Y_train**. И дальше можем сразу посмотреть точность модели с помощью команды ***score***.



Точность получилась 83,3%, что выше предыдущих моделей: метода опорных векторов (76,3%), дерева решений (81,7%) и бэггинга (80%).

Таким образом, на что хотелось бы обратить внимание. То, что задача специалистов, которые занимаются машинным обучением сегодня состоит в том, чтобы как раз подобрать такую модель, которая бы смогла предсказывать с максимальной точностью. Особых глубоких математических знаний иметь необязательно, необходимо просто представлять, как работают разные модели и для каких задач лучше использовать тот или иной алгоритм. И второй момент, как вы сами видели, само обучение модели занимает не так много времени, да, у нас была малая выборка, но даже если в нашем датасэте десятки и сотни тысяч объектов, то само обучение проходит относительно не так долго. И всего несколько строк кода. Что занимает больше всего времени – так это приведение наших данных в порядок, чтобы они были готовы к использованию. Ну и конечно же понимание как эти данные взаимодействуют между собой, а для этого вам необходимо будет в том числе иметь хотя бы минимальное представление о той отрасли, где вы работаете, будь то телекоммуникации, ритейл, нефтедобыча или что-то иное.

На этом пока все, тренируйтесь составлять разные модели немного изменяя датасэты, либо скачивая их в свободном доступе, например, на сайте kaggle.

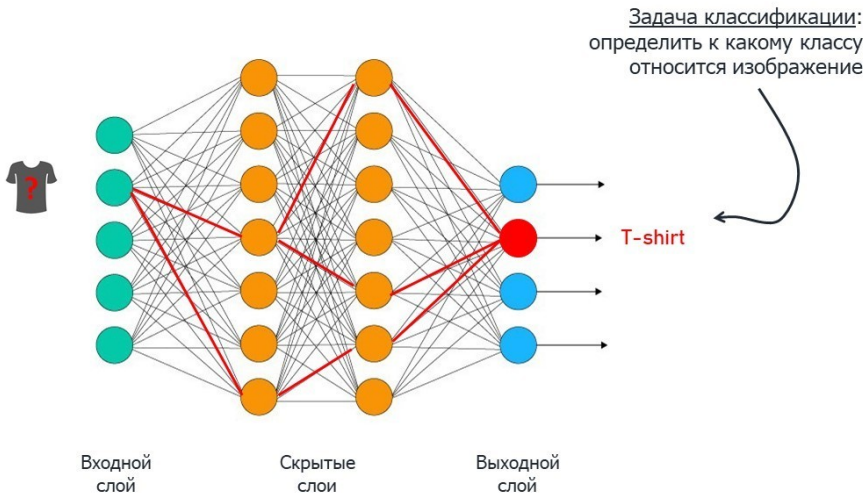
Нейронные сети. Распознавание изображений

Итак, мы переходим к последней важной главе в этой книге. И она будет посвящена нейронным сетям. К концу этой главы вы построите свою собственную нейронную сеть, которая сможет распознавать изображения одежды.

Для этой задачи мы будем использовать набор данных, который называется Fashion MNIST. Он содержит 70 000 изображений различных видов одежды, таких как футболки, брюки, туфли, сумки, свитера, пальто, кроссовки и так далее. С этого датасэта начинает свое обучение любой специалист по глубокому обучению, он является так называемым стандартом для обучения специалистов по нейронным сетям.

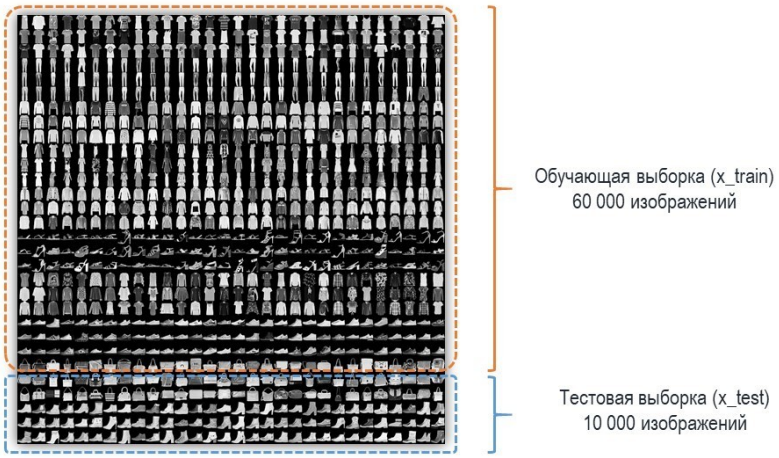
Напомним, что нейронные сети состоят из нескольких слоев, во входной слой подаются данные, в нашем случае изображения, хотя это могут быть и звуки, например. Потом эти данные проходят через несколько слоев нейронов, во время такого прохождения, им присваиваются определенные веса, и далее, у нас имеется выходной слой, который должен выдавать ответы, например, что изображено на рисунке.

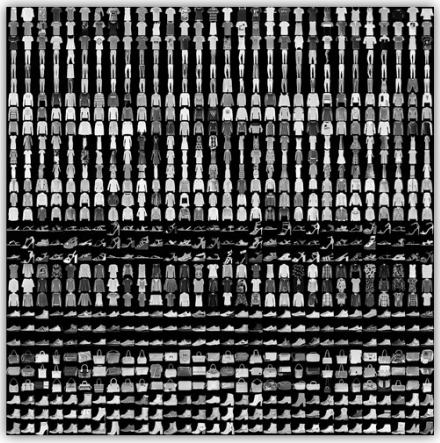
С точки зрения машинного обучения эта задача относится к задачам классификации. То есть на вход нашей нейронной сети подается изображение, а на выходе она должна определить к какому классу одежды относится это изображение, будь то пальто, туфли, футболка или какой-то другой предмет одежды. Всего таких классов в данном наборе 10.



Набор данных Fashion MNIST является открытым, то есть его можно свободно скачать и использовать его для создания нашей нейронной сети. Он был создан компанией Заландо, расположенной в Берлине, и в этот набор входит 70 тысяч различных изображений одежды. В одной из предыдущих глав мы как раз упоминали, что для нейронных сетей, чтобы они хорошо обучились, необходимо как можно больше данных для обучения. Поэтому 70 000 изображений, в принципе, это неплохое количество. Скачать данный набор можно на известном нам сайте github (https://github.com/zalandoresearch/fashion-mnist).

Мы упоминали в прошлых главах, что, когда мы обучаем наши модели, мы делим нашу выборку на две части, обучающую (train) и тестовую (test). Обучающая выборка состоит из 60 000 изображений и ответов (меток) к ним – к какому предмету одежды относится то или иное изображение. Тестовая выборка состоит из 10 000 изображений и будет использована для тестирования, чтобы понять, как хорошо обучилась наша модель.



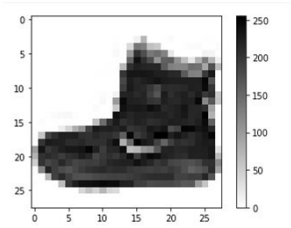


- Футболка/топ
- Брюки
- Свитер
- Платье
- Куртка
- Сандали/туфли
- Рубашка
- Кроссовки
- Сумка
- Ботинки

Изображения (x_train, x_test)

Ответы / метки (y_train, y_test)

Все изображения имеют размер 28 на 28 пикселей и исполнены в оттенках серого. И интенсивность серого цвета в каждом пикселе варьируется по шкале от 0 до 255, где ноль – это полностью белый пиксель, а 255 – это сильный серый цвет. И так как эти картинки такие простые, то есть всего 28 на 28 пикселей и в оттенках серого, это позволяет работать с ними и строить нейронные сети даже если компьютер не слишком мощный.



Каждое изображение размером
28 x 28 пикселей

Интенсивность цвета варьируется
по шкале от 0 до 255

Итак, мы уже сказали, что в наборе Fashion MNIST 10 классов предметов одежды, они пронумерованы от 0 до 9:

0. Футболка/топ
Брюки
Свитер
Платье
Куртка

Сандали/туфли
Рубашка
Кроссовки
Сумка
Ботинки

И, соответственно, когда мы будем подавать в нашу нейронную сеть то или иное изображение, нам будет выдавать номер класса, к которому оно относится. 0 – если это футболка, 5 – если это сандали, и так далее.

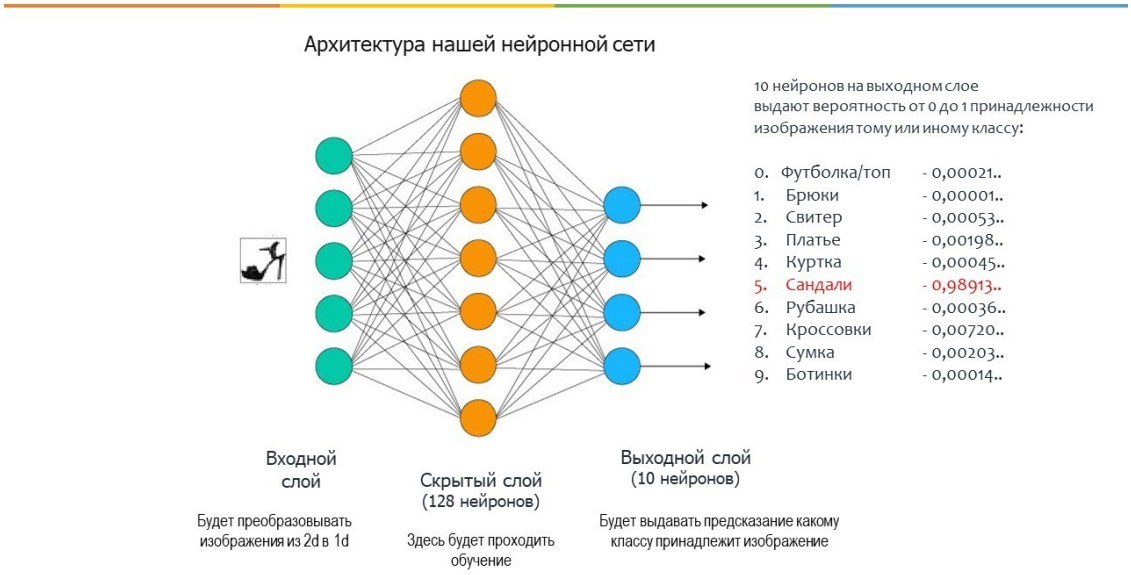
Итак, какие входные данные будут подаваться в нашу нейронную сеть. Как мы уже сказали, во-первых, 28 на 28 пикселей, то есть если перемножить – это 784 пикселей. И во-вторых, это интенсивность оттенков серого в каждом пикселе, интенсивность будет варьироваться от 0 до 255. То есть, все же мы понимаем, что компьютер воспринимает всю информацию в числовом выражении, именно поэтому наша нейронная сеть будет воспринимать каждое изображение как набор 784 пикселей, и каждый пиксель будет иметь конкретное число от 0 до 255 в зависимости от интенсивности оттенков серого.

Архитектура нейронной сети

Идем дальше. Архитектура нашей нейронной сети будет очень простой. Она будет состоять всего из трех слоев, причем первый слой будет просто преобразовывать изображения из двухмерных массивов в одномерные. Следующий полносвязный слой будет состоять из 128 нейронов, и это количество на самом деле, можно изменять: можно сделать, например, 512 или 800 нейронов. И последний выходной слой будет состоять из 10 нейронов по количеству наших классов.

Будет использоваться полносвязная нейронная сеть, а это значит, что все нейроны текущего слоя будут соединены со всеми нейронами предыдущего слоя. На каждый из 128 нейронов основного слоя будут поступать значения всех 784 пикселей изображения.

И каждый из 10 нейронов на выходном слое будет выдавать нам вероятность того, что на изображении данный класс одежды. Вероятность будет отражаться в диапазоне от 0 до 1.



Давайте вспомним, каким образом нейронная сеть может делать такие предсказания и как они вообще работают. Для этого нам пригодятся такие понятия как веса, алгоритм обратного распространения ошибки (на английском back propagation) и эпохи.

Во-первых, как и человеческий нейрон, задача искусственного нейрона получить информацию, обработать ее определенным образом и передать дальше в следующий нейрон. И соединения таких нейронов в искусственном интеллекте, которые получают входные данные, обрабатывают их, и далее выводят выходные данные, такие структуры и называются нейронными сетями.

Так вот, мы с вами уже поняли, что на каждый нейрон поступает изображение в формате 784 пикселей, и у каждого пикселя имеется числовое значение в зависимости от интенсивности цвета. То есть, грубо говоря, на каждый нейрон поступает определенный массив или комбинация числовой информации. Далее, в самый первый раз каждому нейрону или узлу дается какой-то рандомный вес, то есть значимость, или то, насколько это значение нейрона соответствует тому или иному изображению, распределяется случайным образом. Хотелось бы повторить, что на первом этапе этот вес ничем не обоснован. После этого, когда это значение передается на выходной слой, ведь мы же на самом деле знаем, что изображено на

картинке, ведь поэтому же пример классификации называется обучением с учителем, потому что у нас имеются ответы к каждой картинке и мы говорим нейронной сети, либо она угадала со своим предсказанием, либо нет. Так вот, если сеть угадала с ответом, то вес этого значения или этой связи между нейронами в разных слоях увеличивается. А если не угадала, этот вес снижается, и это и называется алгоритм обратного распространения ошибки, то есть мы обратно отправляем информацию в нейронные слои и говорим либо увеличить либо уменьшить вес в зависимости от того, правильно или нет было сделано предсказание. И это движение в сторону изменения весов происходит постоянно во время обучения вперед и назад. Когда весь набор данных, то есть изображений, пройдет через эту процедуру, это называется одна эпоха. И во время обучения нейронных сетей обычно используют несколько эпох, например 10 или даже 100. То есть 100 раз прогоняют одни и те же изображения через все слои, каждый раз изменяя веса нейронов и делая их все более точными, снижая таким образом ошибку предсказания.

Подготовка к созданию. Библиотека

Tensorflow

Итак, давайте посмотрим, как можно создать и обучить такую нейронную сеть в Keras в Tensorflow – это библиотеки в Питоне, которые как раз предназначены для работы с нейронными сетями.

Сразу хочу оговориться, что данный проект мы будем решать в Google Colab. Мы уже упоминали в одной из первых лекций по Питону, что это облачная среда от Google, которая предоставляет возможность тестировать проекты по машинному обучению прямо в браузере. Почему мы выбираем Google Colab сейчас? Потому что для решения этой задачи с нейронными сетями, нам понадобится библиотека Tensorflow и модуль Keras, а если вы хотите установить их у себя на компьютере, это очень часто бывает проблематично, и возникает очень много ошибок с различными версиями, или совместимостью с видеокартой и другие ошибки. А в Google Colab ничего устанавливать не надо, вы просто создаете ноутбук и сразу начинаете кодить. Поэтому в принципе если хотите, вы можете попробовать загрузить Tensorflow на свой компьютер, делается это с помощью команды pip install tensorflow в командной строке Анаконды (в черном окошке), но с очень большой вероятностью у вас могут возникнуть трудности с дальнейшим импортированием этих модулей в ваш проект. И если такие ошибки с установкой возникнут, вот по этой ссылке можете посмотреть самые типичные из них, и как их можно устранить (https://www.tensorflow.org/install/errors).

Поэтому, чтобы избежать этих трудностей с установкой Tensorflow на локальном компьютере, мы покажем как это делать в Google Colab, и здесь уж точно у вас не должно возникнуть никаких проблем. Кроме того, эта облачная среда от Google дает вам возможность использовать больше оперативной памяти, чем возможно имеется на вашем компьютере.

Итак, давайте создадим новый файл (в Google Colab он именуется блокнот) в Google Colab (https://colab.research.google.com/).

Теперь, для начала нам необходимо подключить нужные нам модули.

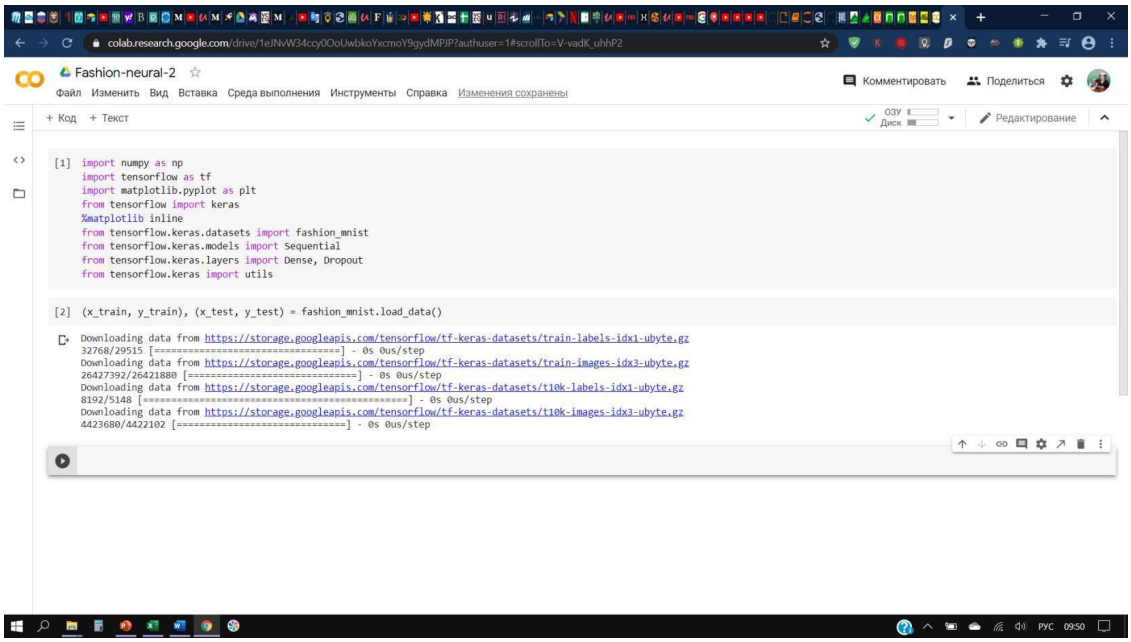
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
%matplotlib inline
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import utils
```

Модуль для работы Fashion_MNIST уже присутствует в Keras, потому что он является одним из таких популярных наборов данных, на которых обучаются нейронным сетям все дата сайентисты. Мы также импортируем модель Sequential – это модель нейронных сетей, где слои идут друг за другом. Далее подключаем тип слоев Dense, который означает что наши слои будут полносвязными. Ну и также подключаем различные утилиты, которые помогут перевести наши данные в подходящий для Keras формат. И напоследок подключим Numpy и Matplotlib для работы с массивами и визуализации наших рисунков.

Так как мы уже импортировали набор данных Fashion_MNIST в первой строчке, мы теперь можем прямо здесь в проекте загрузить их следующей строкой:

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

Как мы уже говорили в предыдущих главах, при машинном обучении наши наборы данных будут разделяться на две части – обучающую и тестовую, на английском train and test. И каждая из этих частей будет содержать часть X – это изображения в нашем случае, и часть Y – это ответы к какому классу принадлежит то или иное изображение или, как они еще называются, метки.



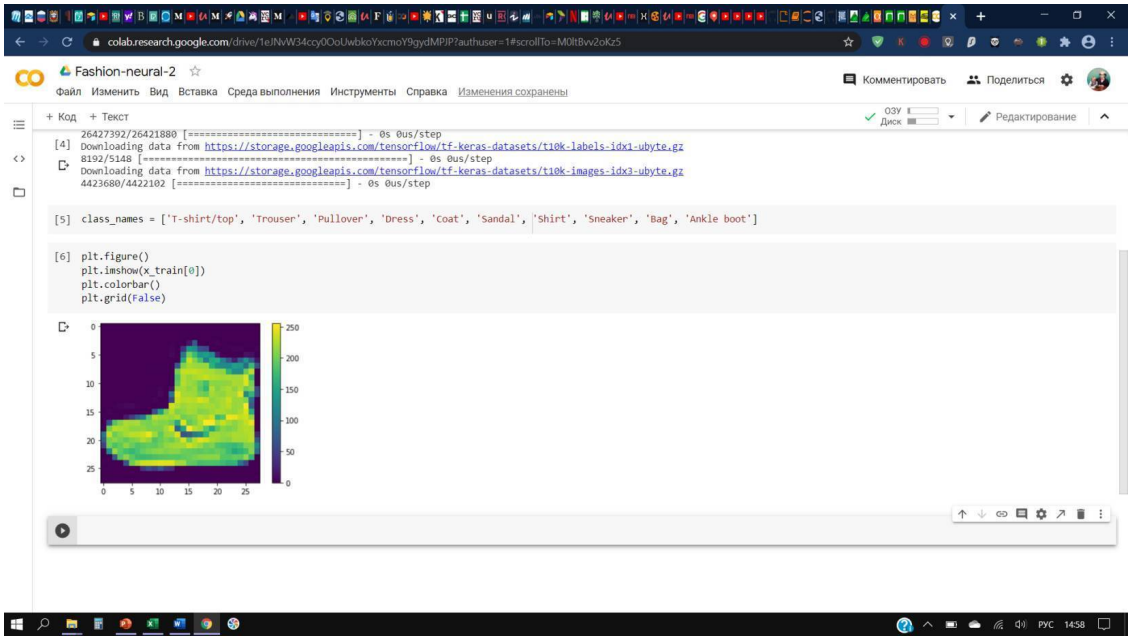
Имена классов не включены в набор данных, поэтому давайте пропишем их сами:

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

Сейчас нам надо провести предварительную обработку наших данных перед тем как создать нейронную сеть. Но давайте сначала посмотрим, как выглядят наши изображения:

plt.figure()
plt.imshow(x_train[0])
plt.colorbar()
plt.grid(False)

Вот в эти квадратные скобки мы вставляем индекс от нуля до 59999, потому что у нас 60000 рисунков в обучающей выборке, и нам будут выдавать изображения. Некоторые рисунки достаточно легко понять, а на некоторых практически непонятно что изображено.



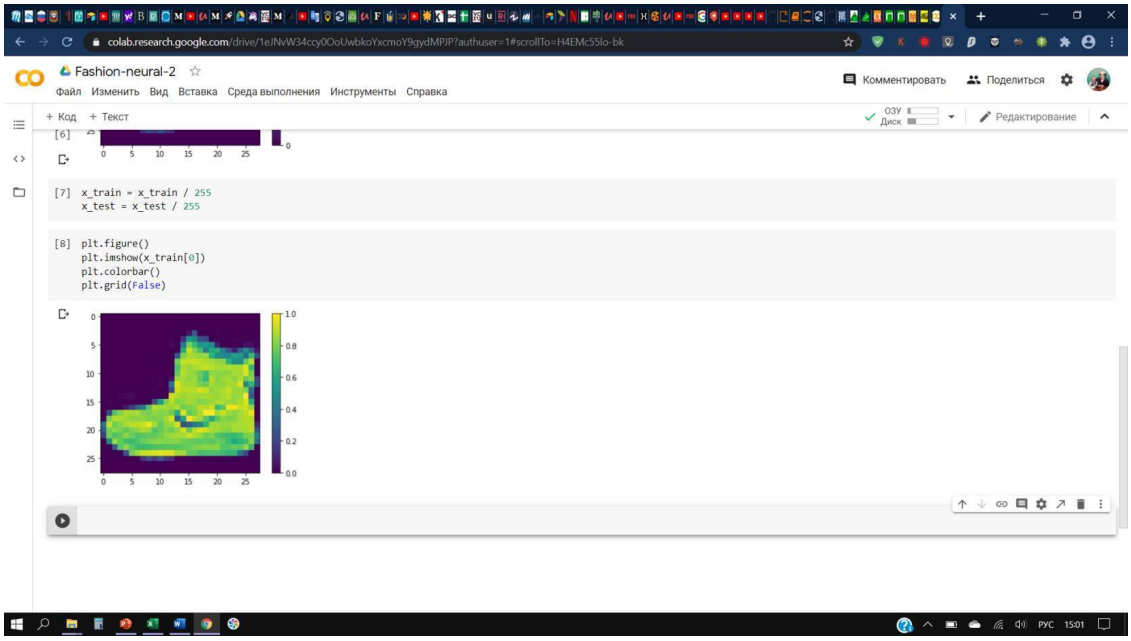
Сбоку вы также можете увидеть значения интенсивности пикселей от 0 до 255. То есть если пиксель максимально темный, он имеет значение 0, а если он максимально светлый, он имеет значение 255.

Давайте сделаем небольшую **нормализацию данных**. Что это значит? Это значит, что чтобы улучшить алгоритмы оптимизации, которые используются в обучении нейронных сетей, мы делим интенсивность каждого пикселя в изображении на 255, чтобы данные на входе в нейронную сеть находились в диапазоне от 0 до 1. И важно, чтобы тренировочный сет и проверочный сет были предобработаны одинаково:

x_train = x_train / 255
x_test = x_test / 255

Давайте проверим.

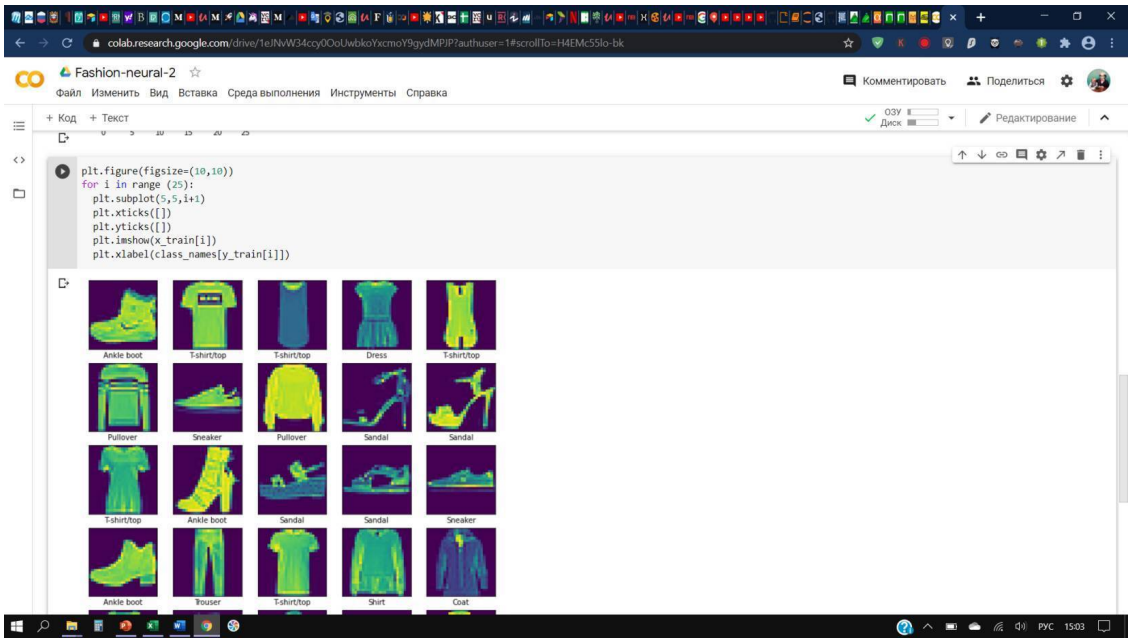
plt.figure()
plt.imshow(x_train[0])
plt.colorbar()
plt.grid(False)



Отлично, теперь интенсивность пикселей от 0 до 1, и нейронным сетям будет проще работать с такими значениями.

Мы можем, кстати, посмотреть сразу несколько изображений на одном экране. Для этого потребуется написать несколько строчек кода. Покажем первые 25 рисунков по 5 в каждой строке. и отобразим под ними наименования их классов.

```
plt.figure(figsize=(10,10))
for i in range (25):
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(x_train[i])
  plt.xlabel(class_names[y_train[i]])
```



Отлично, если хотите, можете сделать рисунки черно-белыми, добавив команду **cmap=plt.cm.binary**.

plt.imshow(x_train[i], cmap=plt.cm.binary)

Создание модели нейронной сети

Итак, данные мы подготовили, теперь мы можем начать создавать нейронную сеть. В нейронных сетях основным строительным блоком является слой. И основная часть глубокого обучения состоит в объединении простых слоев.

В Keras нейронные сети, как и в целом в машинном обучении, называются моделями. И как уже говорили, мы будем использовать модель типа Sequential, что означает, что наши слои будут идти последовательно.

Итак, создаем последовательную модель.

```
model = keras.Sequential([
                keras.layers.Flatten(input_shape=(28,28)),
                keras.layers.Dense(128, activation="relu"),
                keras.layers.Dense(10, activation="softmax")
])
```

Первый слой сети, который указан как flatten, преобразует формат изображений из 2-мерного массива (где каждое изображение являлось массивом 28 на 28 пикселей) в одномерный массив 28 на 28, то есть теперь изображение будет поступать в нейрон, грубо говоря, как строка в 784 пикселя.

Далее уже идут два полноценных слоя. Первый слой – входной полносвязный. Здесь мы должны решить сколько у нас будет нейронов в этом слое, на самом деле по этому набору данных проводилось уже много экспериментов, и одним из самых удачных по предсказанию оказался входной слой по 128 нейронов, хотя, если захотите, можно сделать и 512 или 800 нейронов, на ваше усмотрение.

Дальше пишем функцию активации, в нашем случае для входного слоя мы указываем relu, она показала хорошую эффективность в таких простых нейронных сетях. И далее второй слой – выходной, полносвязный. В нем будет 10 нейронов по количеству наших классов. В качестве функции активации будет функция softmax, благодаря которой второй слой будет возвращать нам массив из десяти вероятностных оценок, сумма которых равна 1. Каждый узел или нейрон будет содержать оценку, которая указывает вероятность того, что текущее изображение принадлежит одному из 10 классов.

Компиляция модели

Перед обучением модели надо сделать еще небольшие настройки, это называется скомпилировать модель. При компиляции модели мы указываем параметры обучения.

model.compile(optimizer=tf.keras.optimizers.SGD(), loss='sparse_categorical_crossentropy', n

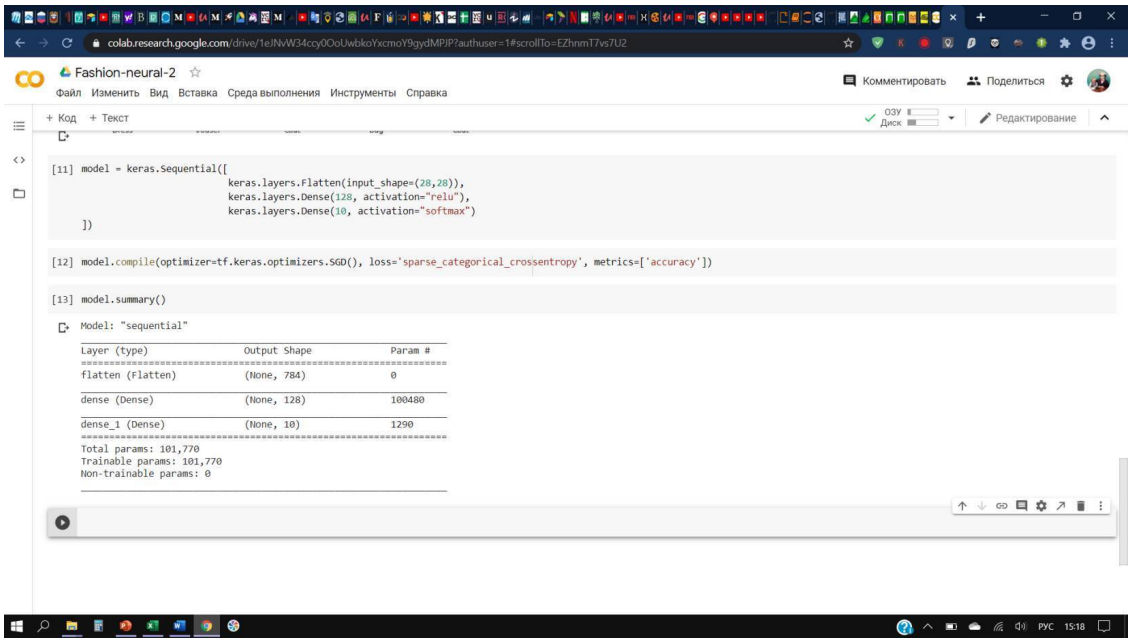
Оптимизатор – в нашем случае мы используем SGD, который расшифровывается как стохастический градиентный спуск. Данный оптимизатор очень популярен для решения подобных задач распознавания изображений с помощью нейронных сетей. Как вариант, можно еще использовать оптимизатор Adam.

Функция ошибки, указанная параметром loss – в нашей модели мы будем использовать не среднеквадратическое отклонение, а категориальную перекрестную энтропию. Данная функция ошибки хорошо работает в задачах классификации, когда классов больше двух.

И последний параметр – параметр качества, мы указываем accuracy – то есть доля правильных ответов.

После того как модель скомпилирована, мы можем напечатать ее параметры с помощью:

model.summary()



В принципе все готово, все данные подготовлены, мы создали и скомпилировали нейронную сеть. Теперь можем приступить к ее обучению.

Обучение модели

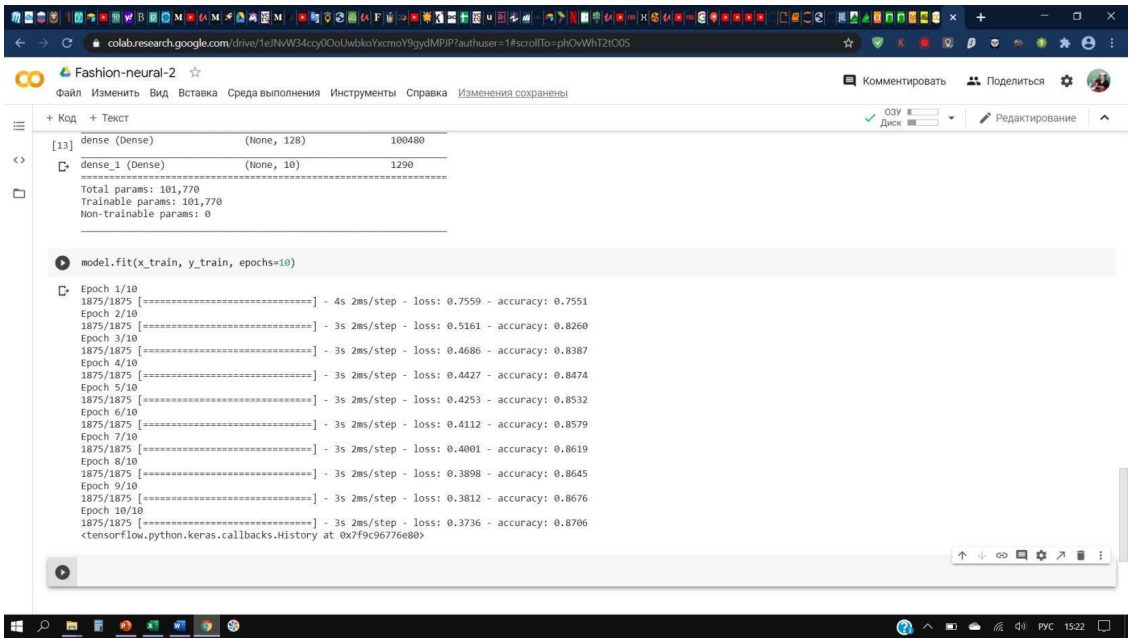
Обучение выполняется так же, как и в других задачах машинного обучения с помощью функции **fit**. И так как у нас обучение с учителем, мы передаем этой функции как обучающую выборку **x_train**, так и ответы **y_train**.

model.fit(x_train, y_train, epochs=10)

Мы также должны указать параметр – количество эпох. Одна эпоха – это когда наш весь набор данных проходит через нейронную сеть один раз. Мы указываем 10 эпох, то есть мы 10 раз будем обучать нашу нейронную сеть на всем наборе наших данных, то есть на всех 60 000 картинках.

Сколько нужно эпох? Ответ будет разниться в зависимости от разных наборов данных. Но основной принцип – это чем больше разношерстный наш датасэт, тем желательно использовать больше эпох. Ну и не забывайте мощности вашего компьютера. Если у вас очень большое количество данных, и они все очень различны, то прогон каждой эпохи будет занимать больше времени.

Поэтому для быстроты обучения, ограничимся 10-ю эпохами. Вы самостоятельно можете поставить большее количество эпох, просто это займет больше времени, а качество будет скорее всего ненамного лучше. Итак, мы видим как началось обучение. В конце строки каждой эпохи указывается функция ошибки через параметр **loss** и точность предсказаний **accuracy**.

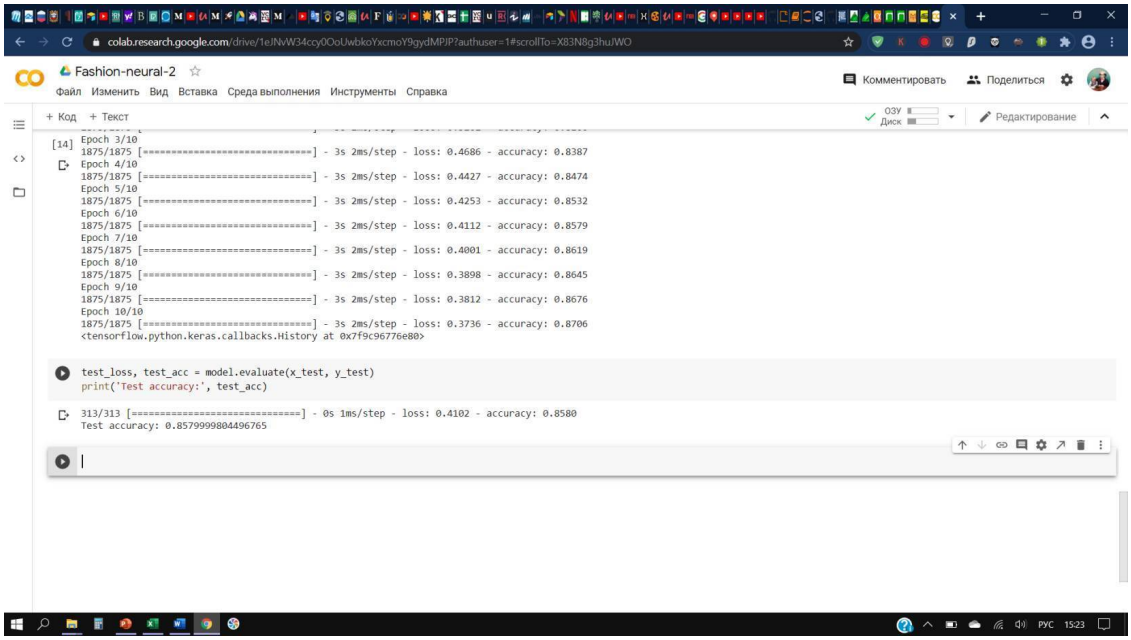


Как можем заметить по мере обучения нашей нейронной сети, то есть с каждой следующей эпохой, значение ошибки снижается, а точность, наоборот, повышается. Итак, вот закончилась последняя эпоха, и это значит, что обучение нашей нейронной сети тоже завершено. Мы видим, что точность составляет около 87%. В целом, для нейронной сети, состоящей всего из 2 слоев, такое качество вполне хорошее.

Давайте теперь проверим какая у нас точность на тестовой выборке. Помните, что для обучения мы использовали 60000 изображений, а 10000 изображений были в тестовой выборке, и наша нейронная сеть их не видела. Поэтому посмотрим, какая будет точность предсказания на новых для нейронной сети изображениях.

test_loss, test_acc = model.evaluate(x_test, y_test)

```
print('Test accuracy:', test_acc)
```



Как видим качество предсказания (85,8%) немного ниже, чем на обучающей выборке, но все равно относительно хорошее.

Предсказание изображений

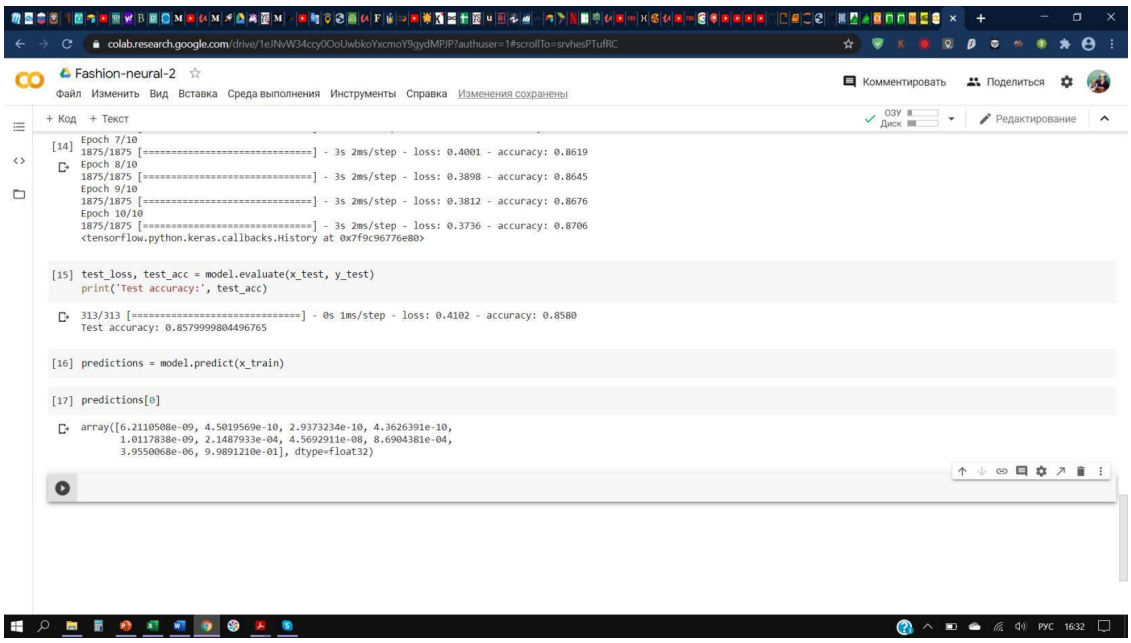
Итак, теперь после завершения обучения мы можем использовать нашу нейронную сеть чтобы предсказывать, что изображено на рисунках.

Для этого используем метод **predict**. Будем предсказывать на тех изображениях, на которых наша модель обучалась, поэтому в скобках запишем **x_train**.

predictions = model.predict(x_train)

И теперь давайте выведем на печать то, что предскажет наша модель по изображению с индексом пусть будет 0.

predictions[0]



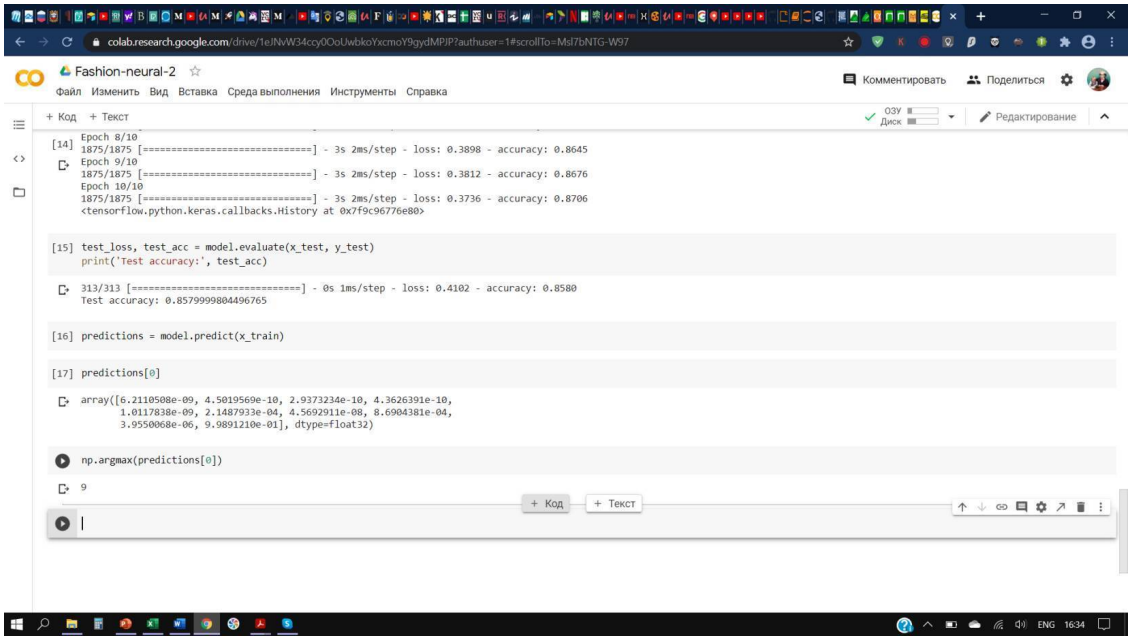
Вот мы видим 10 значений по количеству наших нейронов и по количеству наших классов.

У каждого числа в конце стоит минус 10, либо минус 11. Это означает, что они в -10 степени, то есть после нуля мы имеем еще несколько нулей, и то есть вероятность близка к нулю. И только одно число со значением в минус первой степени, вы видите девять целых девяносто восемь в минус первой, это означает примерно 0 целых 99 сотых, то есть очень близко к 1, а единица в нашем случае – это 100% вероятность. Таким образом наша модель с вероятностью в почти 100% предсказывает, что это изображение соответствует последнему классу.

Если мы не хотим затрудняться и искать, какое из чисел из наших выходных данных максимальное, мы можем воспользоваться функцией **argmax** из библиотеки **Numpy**, она как раз выдает максимальное значение.

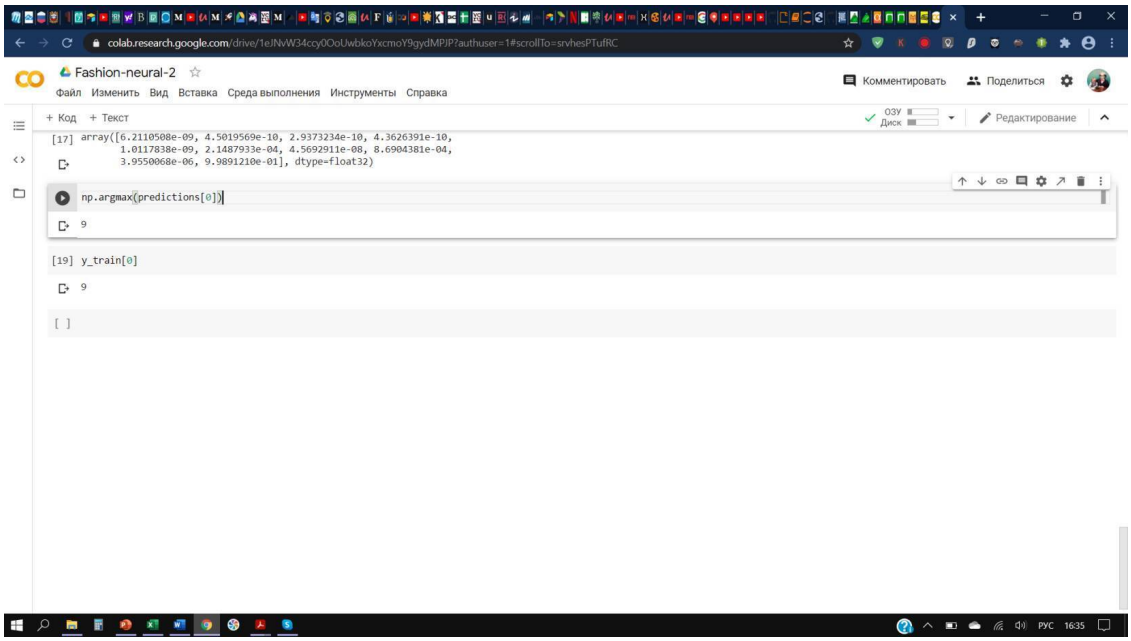
np.argmax(predictions[0])

И вот нам выдали индекс максимального значения, это 9.



Теперь давайте проверим и выведем правильный ответ из наших меток.

y_train[0]



Итак, наш реальный ответ по этому изображению оказался таким же, как и был предсказан нашей моделью. Значит наша модель работает.

Вы можете протестировать с другими изображениями. Просто замените число в индексе на любое другое число из нашей выборки в 60 000 изображений.

Давайте протестируем номер 12.

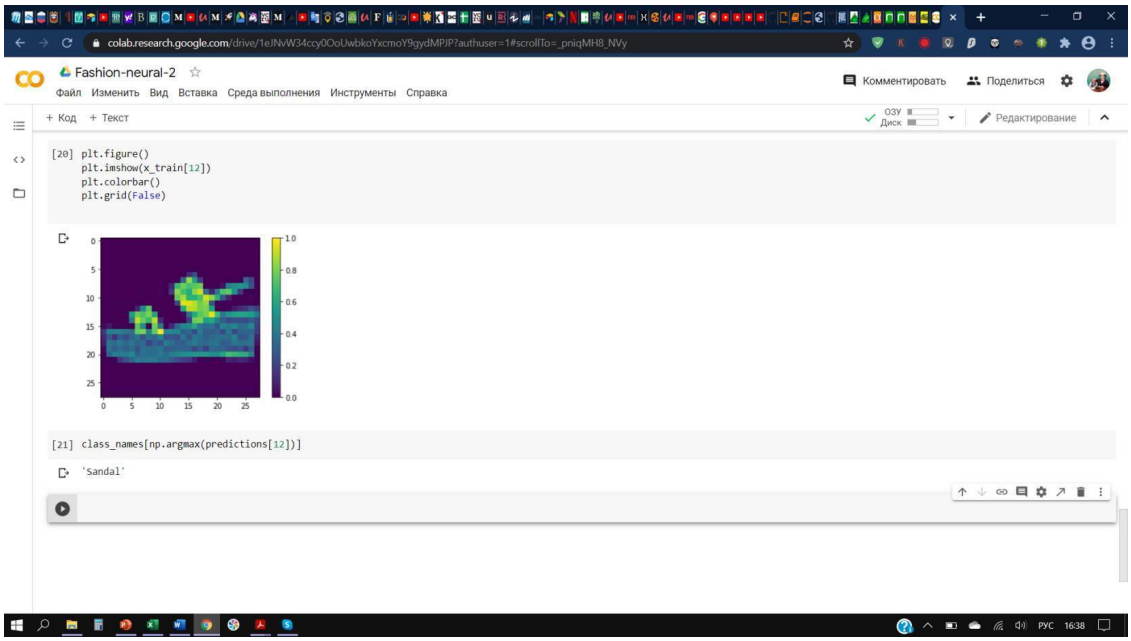
Ок, нам выдает, что это класс с индексом 5. Давайте посмотрим на рисунке, что это. Это сандалии.

plt.figure()
plt.imshow(x_train[12])
plt.colorbar()

plt.grid(False)

И в принципе, если не хотите каждый раз смотреть через рисунки, можем просто напечатать название класса, вставив соответствующий индекс изображения.

class_names[np.argmax(predictions[12])]



Итак, если суммировать, то что мы сделали. Мы импортировали изображения, поделили их на обучающую и тестовую выборку, далее мы немного оптимизировали эти изображения. После этого мы создали архитектуру нейронной сети, которая в нашем случае состояла всего из трех слоев, скомпилировали ее, то есть указали параметры обучения, далее мы обучили нашу нейронную сеть с помощью нашей обучающей выборки и, наконец, протестировали ее на нашей тестовой выборке и проверили точность предсказания.

Попробуйте предсказать другие изображения в обучающей и тестовой выборке, вы можете также изменить модель, например в качестве оптимизатора можете использовать другой, например Adam, или изменить количество эпох и так далее.

Кстати, на известном нам уже сайте Kaggle есть различные варианты нейронных сетей других видов с разным количеством слоев и различными архитектурами для проекта Fashion MNIST. Поэтому можете посмотреть и попробовать сделать другую нейронную сеть для этого датасэта.

https://www.kaggle.com/danialk/range-of-cnns-on-fashion-mnist-dataset

Надеюсь, эта глава была полезна, и Вы пришли к лучшему пониманию, что такое нейронные сети, как их обучать и использовать в том числе для распознавания изображений.

Бонусная глава. Открытые датасэты для задач машинного обучения

Поздравляю вас с завершением данной книги. Ее целью было познакомить вас с основными понятиями в области искусственного интеллекта и машинного обучения. Изначально книга состояла только из теоретической части, но как дополнение мы также включили в нее небольшой блок по основам программирования на Питоне. Потому что именно на этом языке сегодня решаются большинство задач машинного обучения.

На чем хотелось бы еще раз остановиться. Это то, что это был базовый курс и, естественно, в нем мы не могли осветить все тонкости машинного обучения. Как вы понимаете, это очень широкая область знаний, и поэтому рекомендуем вам продолжать обучение с помощью дополнительных более специфических курсов и книг именно по той тематике и отрасли, которую вы для себя выбрали и которая вам необходима в конкретный момент времени.

А в данной заключительной главе, как небольшой полезный бонус, хотели бы поделиться с вами ресурсами, где можно найти в свободном доступе различные датасэты, которые вы можете использовать для тренировки своих моделей и для практики своих знаний в машинном обучении.

Как вы понимаете, все машинное обучение строится на данных и причем чем они объемнее и более подготовленные, тем более эффективным будет машинное обучение и точность моделей.

Итак, вот список этих ресурсов, где вы можете найти большое количество бесплатных и свободных для скачивания датасэтов для самостоятельной практики.

Kaggle

https://www.kaggle.com/datasets

Начнем с Kaggle. На данный момент здесь собрано около 30 000 различных датасэтов на различную тематику. Их можно найти с помощью поисковой строки, причем можно кликнуть на кнопку Preview, чтобы посмотреть краткую информацию по каждому датасэту, не открывая его полностью. Кроме того, другим преимуществом Kaggle, помимо его обширной библиотеки датасэтов, является то, что к практически каждому датасэту прикреплены фрагменты кода других пользователей, которые работали с ним. Есть возможность пообщаться с другими членами этого сообщества по тому или иному датасэту во вкладке дискуссии. Часто также прикреплены ссылки на различные соревнования по тем или иным датасэтам, чтобы вы могли посмотреть какую точность показывают модели других участников.

UCI

https://archive.ics.uci.edu/ml/index.php

Следующий ресурс это UCI Machine Learning Repository, который содержит около 500 различных датасэтов для машинного обучения и анализа данных. Датасэты помечены категориями, к какому основному виду задач они относятся, будь то классификация, регрессия или рекомендательные системы. Также имеется поисковая строка, чтобы удобнее было найти нужный датасэт. Все датасэты из реальных проектов, что поможет вам понять с какими типами задач вы будете сталкиваться, когда будете работать над реальным проектом уже самостоятельно в той или иной компании.

Scikit-learn

https://scikit-learn.org/stable/datasets/index.html

Нельзя не упомянуть о датасэтах из самой библиотеки scikit learn, которую мы использовали в данном курсе, чтобы работать с задачами регресии и классификации. Датасэты, содержащиеся здесь, являются как из реальных проектов, так и специально созданные для обуче-

ния. Их очень легко сразу загружать в ваш проект, с помощью пары строк кода. Например, если вы хотите посмотреть цены на квартиры в Бостоне и отчего они зависят, можете просто воспользоваться следующим кодом.

```
from sklearn.datasets import load_boston
boston = load_boston()
```

Drivendata.org

Еще один ресурс drivendata.org устраивает различные соревнования по машинному обучению. Проекты на данном сайте посвящены в основном решению социальных проблем. Например, предсказать распространение инфекций, или предсказать возможный ущерб зданиям от землетрясения в зависимости от расположения и типа конструкций, или распознать животных, которые были захвачены на камеру ловушку в одном из национальных парков в Африке. Соревнования очень интересные и полезные для общества в целом и за победу в них можно получить неплохое денежное вознаграждение.

Visualdata.io

Если вы работаете над задачами, связанными с обработкой изображений, компьютерным зрением или глубоким обучением, то еще одним полезным ресурсом станет сайт visualdata.io На данный момент он содержит более 400 датасэтов. Тут есть датасэты, которые помогут вам в создании моделей для автономного вождения, или для того, чтобы, например, рассчитать количество людей в очень тесной толпе или на масштабном мероприятии, или распознать какая еда находится на тарелках людей во время известного фестиваля еды.

Microsoft, **Amazon**, **Google**

Крупнейшие технологические компании, такие как Microsoft, Amazon, Google, также имеют свои датасэты.

Yahoo.Finance

Если вам необходимы данные по акциям различных компаний или индексам на фондовых биржах, то можно найти эту информацию для скачивания на Yahoo.Finance.

Данные государственных органов и международных организаций

И конечно же не забывайте, что международные организации и страны имеют свои собственные органы статистики, а также многие министерства, в том числе, экономики, финансов, центральные банки, также регулярно выпускают статистическую информацию по тем или иным соответствующим показателям. Поэтому вы также можете искать нужные вам данные из этих официальных источников.

Надеюсь, эта книга была вам полезна, желаю вам успехов в машинном обучении и не только, и до встречи!

(Все изображения в книге являются художественными работами и принтскринами автора)