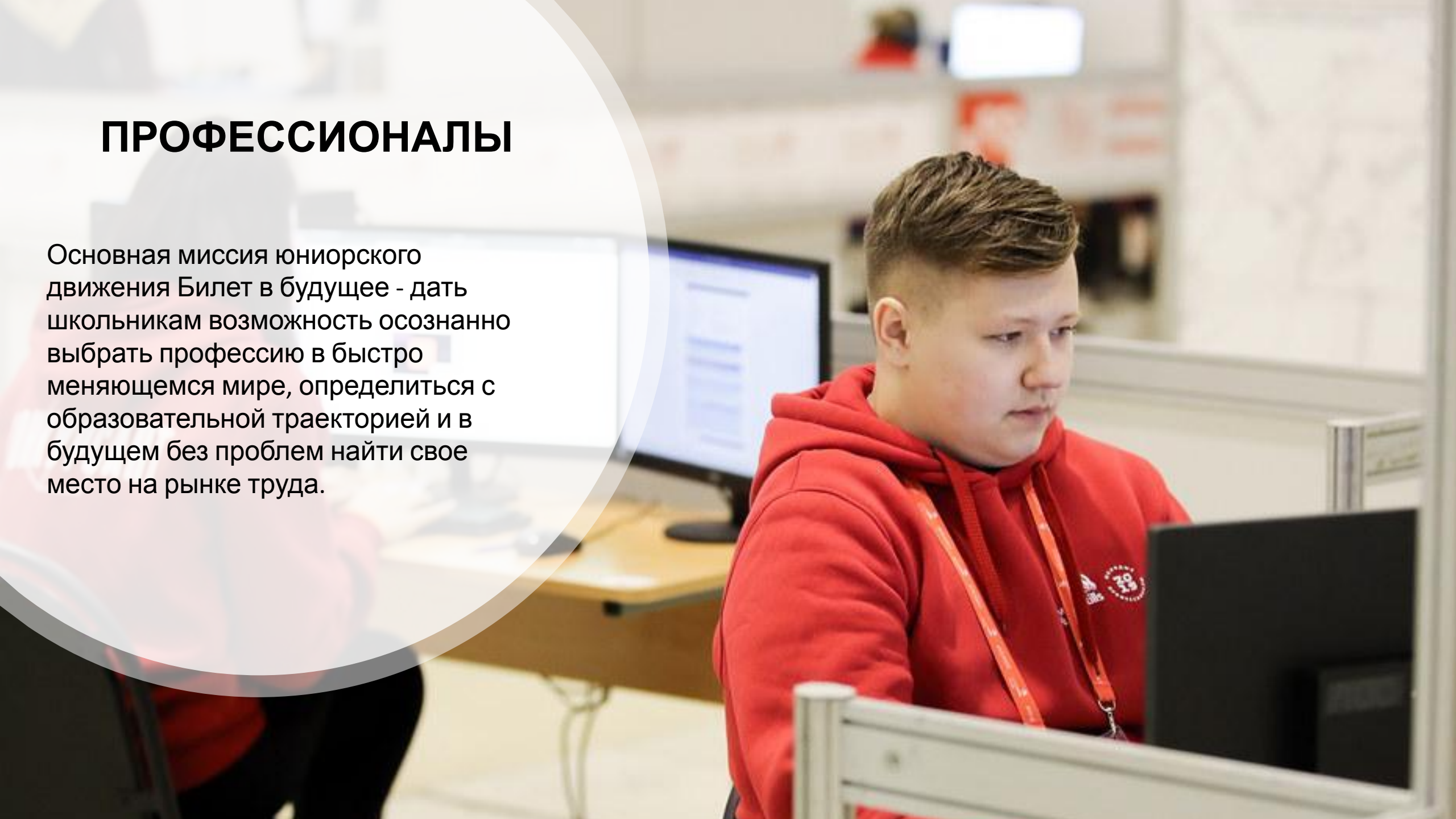
Машинное обучение и большие данные

ПРОФЕССИОНАЛЫ

Основная миссия юниорского движения Билет в будущее - дать школьникам возможность осознанно выбрать профессию в быстро меняющемся мире, определиться с образовательной траекторией и в будущем без проблем найти свое место на рынке труда.



Проект «Билет в будущее»

БИЛЕТ В БУДУЩЕЕ

Проект ранней профессиональной ориентации школьников 6-11 классов:

не менее 100 тыс участников в год получают рекомендации по построению индивидуального учебного плана в рамках выбранного профессионального пути

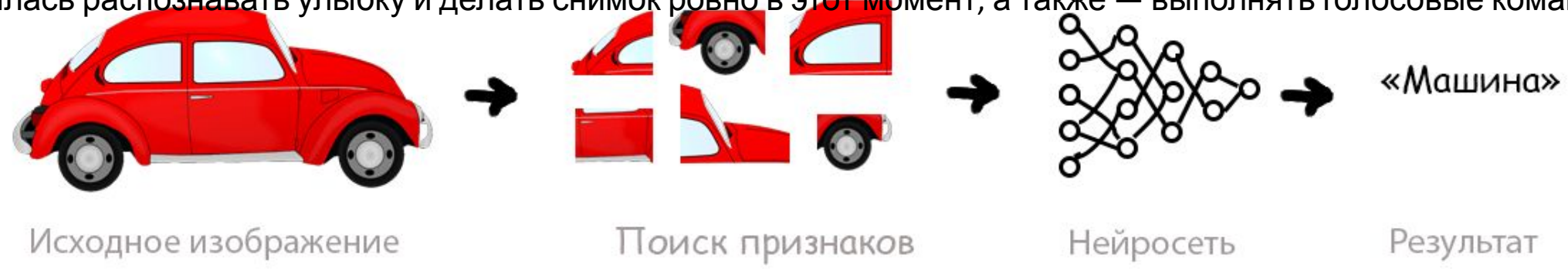
1. Формирование у молодых людей способности строить свою образовательную и карьерную траекторию, осознанно выбирать профессиональный путь.
2. Получение рекомендаций о ближайших шагах в зависимости от уровня осознанности, интересов, способностей школьника, доступных ему возможностей.

Машинное обучение и большие данные

•**Машинное обучение** — это быстроразвивающаяся наука об обработке больших данных, обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться.

За последнее десятилетие машинное обучение было **реализовано** в беспилотных автомобилях, распознавании речи, эффективных поисковых системах и т.д. Его постоянное развитие вызвано ростом возможностей современных вычислительных систем, еще более стремительным ростом объемов данных, доступных для анализа, а также постоянным расширением области применения методов машинного обучения на все более широкий класс задач обработки данных.

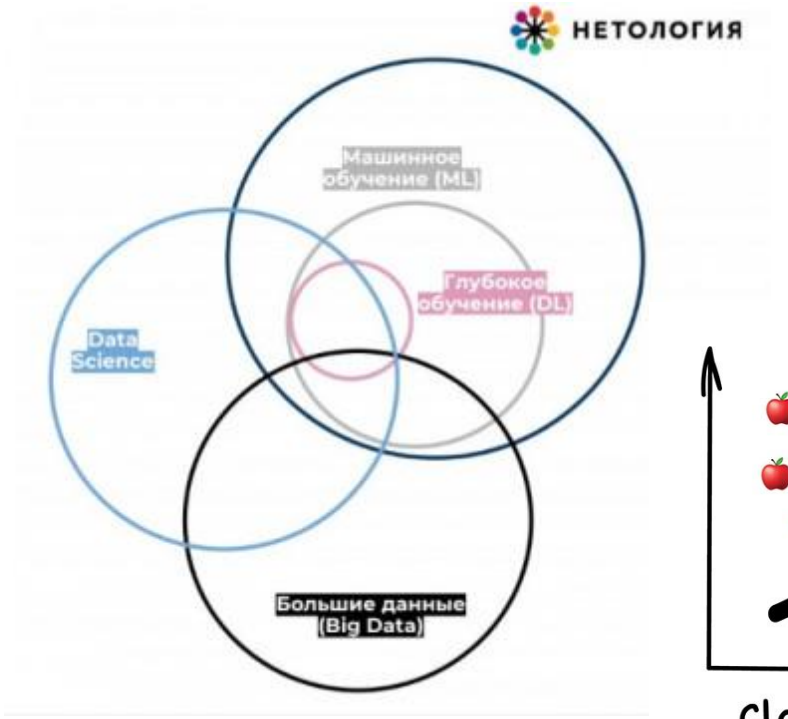
Банки. Программы банковского скоринга решают вопрос с обработкой огромного количества кредитных анкет. Специалисты создают модель, которая автоматически рассчитывает кредитный рейтинг, оценивает платёжеспособность клиента и определяет, одобрить выдачу кредита или отказать в ней.
Маркетинг. Когда Алиса предлагает персональный плейлист, а Yandex показывает персональную ленту, это классическое применение машинного обучения в задаче рекомендации. Другой пример — магазины без касс и продавцов, в которых за счёт машинного обучения алгоритмы учатся соотносить клиента с его виртуальной корзиной и отслеживать перемещения товаров на полках.
Медицина. Один из самых громких примеров — фундаментальное открытие, которое в 2020 году совершил алгоритм AlphaFold. Он смоделировал процесс сворачивания белка, решив одну из самых сложных биохимических задач столетия. Благодаря модели учёные получили возможность предотвращать развитие инфекций, когнитивных и нейродегенеративных заболеваний — Паркинсона, Альцгеймера и других.
Сельское хозяйство. С помощью машинного обучения созданы модели, умеющие анализировать состав почвы, рассчитывать нужное количество удобрений, предсказывать урожайность и даже прогнозировать надои молока у коров.
Гаджеты. Китайский производитель «умных» пылесосов Ecovacs Robotics обучил свои пылесосы распознавать носки, провода и другие посторонние предметы на полу с помощью множества фотографий и машинного обучения. «Умная» камера на базе микрокомпьютера Raspberry Pi 3B+ с помощью фреймворка TensorFlow Light научилась распознавать улыбку и делать снимок ровно в этот момент, а также — выполнять голосовые команды.

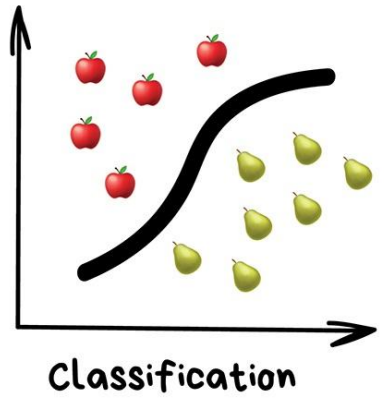


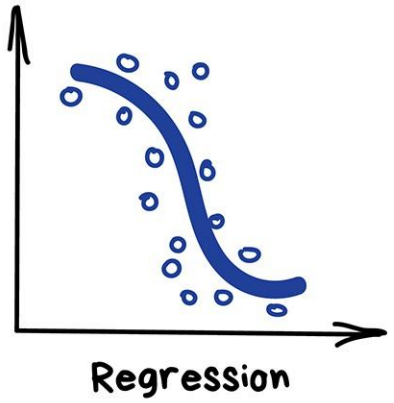
•**Перспективы** развития машинного обучения почти безграничны. С уверенностью можно сказать, что профессия датасайнтиста будет одной из самых востребованных в ближайшем будущем.

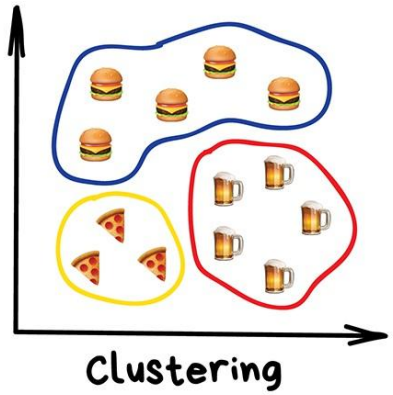
•В рамках компетенции применяются наиболее эффективные алгоритмы машинного обучения, реализуется опыт их практического применения. Рассматривается применение машинного обучения к практическим новым задачам, требующим быстрого и эффективного решения.

- Данная компетенция **формирует навыки** корректной обработки данных, эффективного обмена данными и проведения базовой разведки больших сложных наборов данных, построения и проверки качества моделей, интерпретации математических моделей с целью получения новых нетривиальных знаний и выводов, использования высокоуровневых программных средств для решения типичных задач машинного обучения: кластеризации, классификации, регрессии.



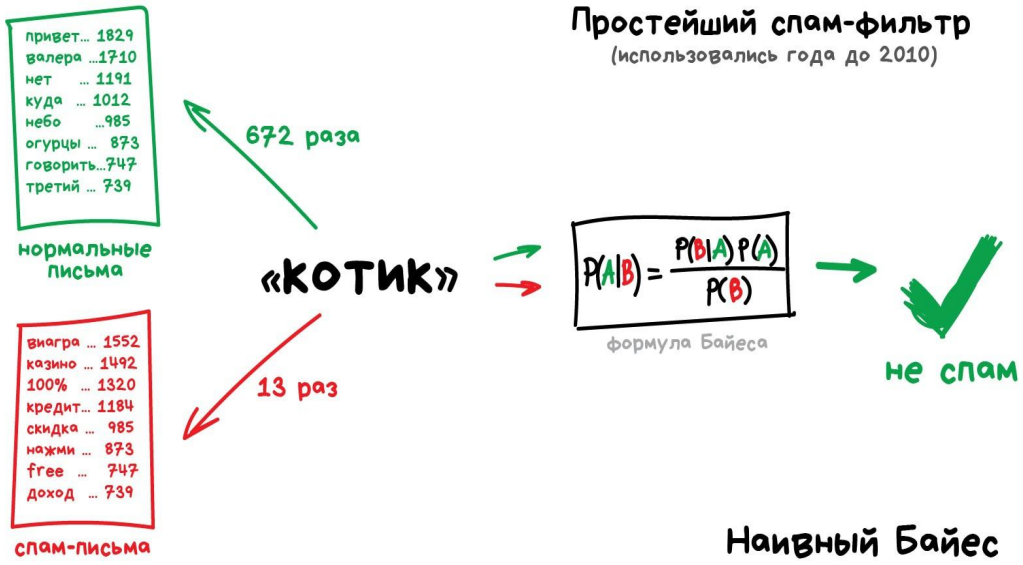






Раньше все спам-фильтры работали на алгоритме Наивного Байеса. Машина считала сколько раз слово «выигрыш» встречается в спаме, а сколько раз в нормальных письмах. Перемножала эти две вероятности по формуле Байеса, складывала результаты всех слов и бац, всем лежать, у нас машинное обучение!

Позже спамеры научились обходить фильтр Байеса, просто вставляя в конец письма много слов с «хорошими» рейтингами. Метод получил ироничное название Отравление Байеса, а фильтровать спам стали другими алгоритмами. Но метод навсегда остался в учебниках как самый простой, красивый и один из первых практически полезных.



Возьмем другой пример полезной классификации. Вот берёте вы кредит в банке. Как банку удостовериться, вернёте вы его или нет? Точно никак, но у банка есть тысячи профилей других людей, которые уже брали кредит до вас. Там указан их возраст, образование, должность, уровень зарплаты и главное — кто из них вернул кредит, а с кем возникли проблемы.

Да, все догадались, где здесь данные и какой надо предсказать результат. Обучим машину, найдём закономерности, получим ответ — вопрос не в этом.

Проблема в том, что банк не может слепо доверять ответу машины, без объяснений. Вдруг сбой, злые хакеры или админ решил *скриптик исправить*.

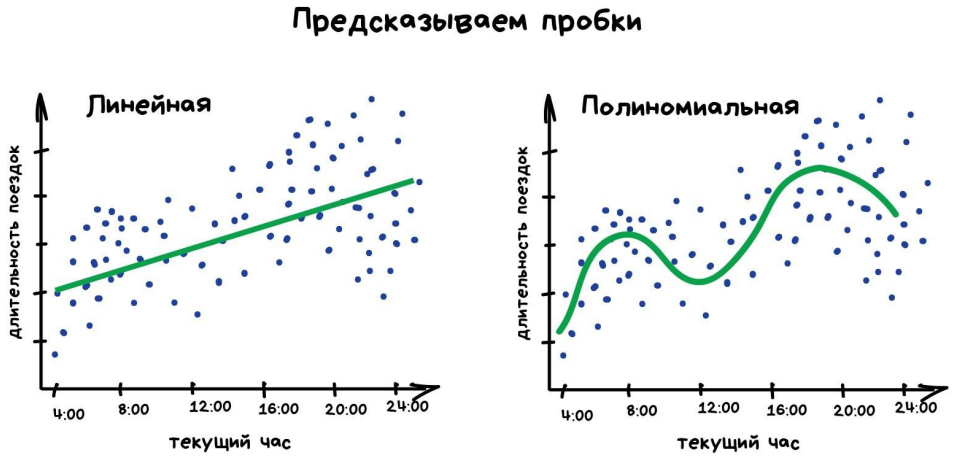
Для этой задачи придумали Деревья Решений. Машина автоматически разделяет все данные по вопросам, ответы на которые «да» или «нет». Вопросы могут быть не совсем адекватными с точки зрения человека, например *«зарплата заёмщика больше, чем 25934 рубля?»*, но машина придумывает их так, чтобы на каждом шаге разбиение было самым точным.

Так получается дерево вопросов. Чем выше уровень, тем более общий вопрос.

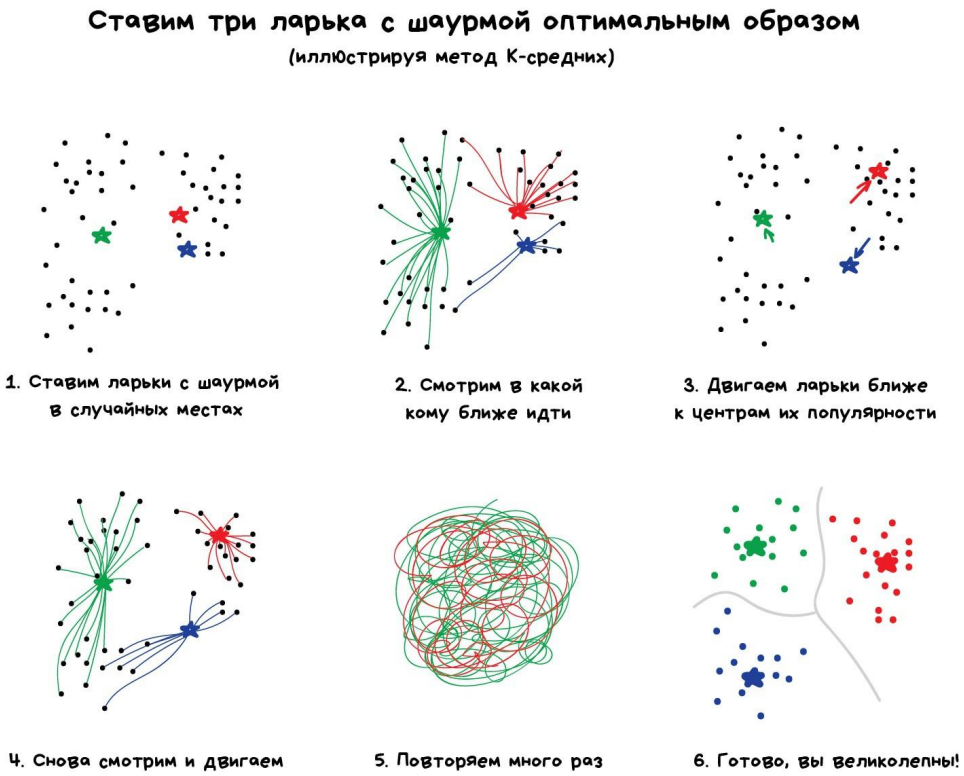
Деревья нашли свою нишу в областях с высокой ответственностью: диагностике, медицине, финансах

Дер

Регрессия — та же классификация, только вместо категории мы предсказываем число. Стоимость автомобиля по его пробегу, количество пробок по времени суток, объем спроса на товар от роста компании и.т.д. На регрессию идеально ложатся любые задачи, где есть зависимость от времени.



Кластеризация — это классификация, но без заранее известных классов. Она сама ищет похожие объекты и объединяет их в кластеры. Количество кластеров можно задать заранее или доверить это машине. Похожесть объектов машина определяет по тем признакам, которые мы ей разметили — у кого много схожих характеристик, тех давай в один класс.



Data Scientist

- **Математическая логика, линейная алгебра и высшая математика.**
- **Знание машинного обучения.** Работа дата-сайентиста — анализ данных огромного размера, и вручную это сделать нереально. Чтобы было проще, они поручают это компьютерам. Поручить такую задачу — значит настроить готовую нейросеть или обучить свою. Поручить программисту обычно это нельзя — слишком много нужно будет объяснить и проконтролировать.
- **Программирование на Python и R.** Python идеальный язык для машинного обучения и нейросетей. На нём можно быстро написать любую модель для первоначальной оценки гипотезы, поиска общих данных или простой аналитики.
- R — язык программирования для статического анализа. Если вам нужно прикинуть, как лайки на странице зависят от количества просмотров или до какого места читатель гарантированно долистывает статью (чтобы поставить туда баннер), — R вам поможет. Но если вы не знаете математику — не поможет.
- **Умение получать и визуализировать данные.** Не всем дата-сайентистам везёт настолько, что они сразу получают готовые наборы данных для обработки. Чаще всего они сами должны выяснить, где, откуда, как и сколько брать данных. Здесь обычные программисты им уже могут помочь — спарсить сайт, выкачать большую базу данных или настроить сбор статистики на сервере.
- Второй важный навык в этой профессии — умение наглядно показать результаты работы. Какой толк в графиках, если никто, кроме автора, не понимает, что там нарисовано? Задача дата-сайентиста — представить данные наглядным образом, чтобы зрителю было легче сделать нужный вывод.

Доход инженеров по обработке данных

- В международной практике начальная зарплата обычно составляет $100 000 в год и значительно увеличивается с опытом, по данным Glassdoor. Кроме того, компании часто предоставляют опционы на акции и 5–15% годовых бонусов.

В России в начале карьеры зарплата обычно не меньше 50 тыс. рублей в регионах и 80 тыс. в Москве. На этом этапе не требуется опыт, кроме пройденного обучения. Через 1–2 года работы — вилка 90–100 тыс. рублей. Вилка увеличивается до 120–160 тыс. через 2–5 лет. Добавляются такие факторы, как специализация прошлых компаний, размер проектов, работа с big data и прочее.

Для сотрудника с опытом работы от 4–5 лет вилка вырастает до 350 тыс.

План работы:

- Постановка задания
- Выполнение задания
- Контроль и оценка
- Открытый курс машинного обучения. https://habrahabr.ru/company/ods/blog/344044/

Задание (машинное обучение): Titanik

В этом задании мы применим полученные знания к датасету Titanik для вот этого соренования: https://www.kaggle.com/c/titanic

Данные скачивать с сайта не нужно, они уже лежат в письме.

Почитать о том, что такое каггл: https://www.kaggle.com/getting-started/44916

Часть 0. Загрузка датасета

Начем!

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

С помощью библиотеки pandas загрузим данные из файла train_titanik.csv в память и выведем первые 5 записей на экран:

In [2]:
```
data = pd.read_csv('train.csv')
data.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

В датасете представлены данные 891 пассажира Титаника. Данные содержат:

-- id пассажира (PassengerId)
-- класс каюты, в которой ехал пассажир (Pclass, 1, 2 или 3)
-- имя пассажира (Name)
-- пол пассажира (Sex)
-- возраст пассажира (Age)
-- количество родственников пассажира на борту Титаника (SibSp)
-- количество родителей/детей пассажира на борту Титаника (Parch)
-- номер билета пассажира (Ticket)
-- цена билета пассажира (Fare)
-- номер каюты пассажира (Cabin)
-- пункт посадки пассажира на Титаник (Embarked)
-- отметка, выжил человек или не выжил после крушения Титаника (Survived) -- **целевая переменная**

Таким образом, нам нужно построить модель, которая по имеющимся данным о пассажире могла бы предсказывать, выжил человек после крушения Титаника или нет.

Как видно, в данных есть пропуски (NAN в колонке Cabin), некоторые данные категориальные (представлены текстом, а не числами). Чтобы на этих данных можно было обучать модели, нужно заполнить пропуски (избавиться от NaN) и перевести категориальные признаки в числовые.

Часть 1. Предобработка признаков

Удалим ненужные колонки

Давайте подумаем, все ли признаки из таблицы нам важны для решения задачи. Выведем еще раз первые строки таблицы для наглядности:

In [3]:
```
data.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Мы, как люди, знаем, что выживаемость на Титанике не зависела от имени пассажира и номера билета. Поэтому давайте удалим эти колонки. Также удалим колонку PassengerId, так как она была добавлена в данные для удобства и, очевидно, выживаемость пассажира не зависла от его ID в этом датасете.

In [4]:
```
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html
data = data.drop(["PassengerId", "Name", "Ticket"], axis=1)
```

Посмотрим на данные

Для начала давайте отделим колонку целевой переменной от датасета и разделим датасет на тренировочный и тестовый. На тренировочном датасете будем обучать KNN, на тестовом -- тестировать.

In [5]:
```
y = data['Survived']
# https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.drop.html
data = data.drop(["Survived"], axis=1)
```

Давайте посмотрим, какие в данных есть колонки и какого типа данные в них записаны:

In [6]:
```
print(data.columns)
print(data.dtypes)
```

```
Index(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Cabin', 'Embarked'], dtype='object')
Pclass        int64
Sex          object
Age         float64
SibSp         int64
Parch         int64
Fare        float64
Cabin        object
Embarked     object
dtype: object
```

И также посмотрим, в каких столбцах данных содержатся NaN:

In [7]:
```
data.columns[data.isna().any()].tolist()
```

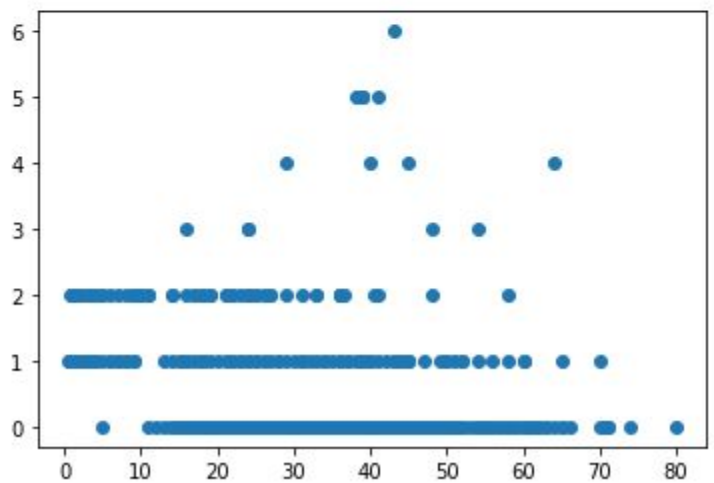
Out[7]:
```
['Age', 'Cabin', 'Embarked']
```

Очень часто бывает полезно посмотреть на связь разных признаков между собой на картинке. Это помогает выявить выбросы (элементы, которые "не списываются" в общую картину и, возможно, были добавлены случайно/на них была допущена ошибка при внесении в датасет или они просто уникальны. Такие данные осложняют обучение моделей) и увидеть неожиданные связи между признаками.

Давайте, например, посмотрим на связь признаков Age и Parch:

In [8]:
```
# https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.scatter.html
plt.scatter(data["Age"], data["Parch"])
```

Out[8]:
```
<matplotlib.collections.PathCollection at 0x2afae5ea730>
```



Заполненим пропуски в данных

Начнем с того, что заполним NaN в колонках, где они есть, какими-нибудь значениями. Для того, чтобы лучше понять, какие значения для пропусков выбрать, полезно посмотреть на данные с точки зрения каких-нибудь статистик (найти среднее по колонке, медиану, самое часто встречающееся значения и т.п.)

1. Age

Возраст человека может быть любым числом от 0 до $\inf$. Поэтому, кажется, что для колонки Age смотреть на количество строк с каждым отдельным значеним не имеет смысла. Больший смысл имеет найти значения статистик.

In [9]:
```
data["Age"].mean(), data["Age"].median()
```

Out[9]:
```
(29.69911764705882, 28.0)
```

In [10]:
```
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html
data["Age"] = data["Age"].fillna(28)
```

2. Embarked

Колонка Embarked, наоборот, содержит категориальный признак, который имеет мало разных значений в датасете. Поэтому для него как раз имеет смысл найти количества записей с каждым значением в датасете.

In [11]:
```
data["Embarked"].value_counts()
```

Out[11]:
```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

Заполним отсутствующие значения самым часто встреющимся значением S:

In [12]:
```
data["Embarked"] = data["Embarked"].fillna('S')
```

Cabin

А вот что делать с колонкой Cabin, сразу непонятно. Она категориальная, и значений в ней много разных.

Давайте посмотрим, сколько всего неизвестных значений в колонке Cabin:

In [13]:
```
data['Cabin'].isna().sum(), len(data)
```

Out[13]:
```
(687, 891)
```

Как видим, пропусков в этой колонке больше, чем имеющихся значений. Это делает заполнение пропусков очень неточным. Учитывая, что сам признак вряд ли очень полезен, давайте эту колонку тоже удалим.

In [14]:
```
data = data.drop("Cabin", axis=1)
```

Отлично, мы разобрались со всеми пропусками, теперь их быть не должно. Давайте убедимся в этом:

In [15]:
```
# выдает ошибку, если в данных остались пропуски
assert not data.isnull().values.any()
```

Перевод категориальных признаков в числа

Модели машинного обучения (за редким исключением) умеют работать только с числовыми признаками. Поэтому все нечисловые признаки придется превратить в числовые.

Еще раз выведем колонки и их типы:

In [16]:
```
print(data.columns)
print(data.dtypes)
```

```
Index(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked'], dtype='object')
Pclass        int64
Sex          object
Age         float64
SibSp         int64
Parch         int64
Fare        float64
Embarked     object
dtype: object
```

У нас всего две категориальные колонки (Sex и Embarked).

In [17]:
```
data['Sex'].describe()
```

Out[17]:
```
count      891
unique       2
top       male
freq       577
Name: Sex, dtype: object
```

1. Sex

В колонке Sex всего два различных значения (male и female). Давайте закодируем эти значения цифрами 0 и 1 и заменим значения в колонке на 0 и 1:

In [18]:
```
data["Sex"] = data["Sex"].astype('category')
data["Sex"] = data["Sex"].cat.codes
```

In [19]:
```
data.head()
```

Out[19]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	1	22.0	1	0	7.2500	S
1	1	0	38.0	1	0	71.2833	C
2	3	0	26.0	0	0	7.9250	S
3	1	0	35.0	1	0	53.1000	S
4	3	1	35.0	0	0	8.0500	S

2. Embarked

In [20]:
```
data['Embarked'].describe()
```

Out[20]:
```
count     891
unique      3
top         S
freq      646
Name: Embarked, dtype: object
```

В колонке Embarked всего 3 уникальных значения. Мы можем поступить с ними так же, как и с колонкой Sex -- заменить значения на цифры 0, 1 и 2. Но это будет не очень хорошо, потому что мы введем для значений в колонке отношение "больше-меньше", которого на самом деле нет.

Решение такое: давайте создадим три столбца, каждый с названием одного из значений колонки. Каждый столбуц будет содержать числа 0 и 1 -- 1 для тех объектов, значение признака Embarked которых соврадает с названием колонки, и 0 -- если не совпадает.

В Pandas для такого есть специальная функция:

In [21]:
```
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html
data = pd.get_dummies(data, columns=["Embarked"])
data.head()
```

Out[21]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S
0	3	1	22.0	1	0	7.2500	0	0	1
1	1	0	38.0	1	0	71.2833	1	0	0
2	3	0	26.0	0	0	7.9250	0	0	1
3	1	0	35.0	1	0	53.1000	0	0	1
4	3	1	35.0	0	0	8.0500	0	0	1

Отлично! Мы завершили предобработку признаков датасета.

Осталось разбить данные на тренировочный и валидационный датасет и можно обучать модель.

Разбиваем данные на train и val

In [22]:
```
from sklearn.model_selection import train_test_split
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
train_data, val_data, train_y, val_y = train_test_split(data, y, test_size=0.3)
```

Часть 2. Обучение модели.

In [23]:
```
from sklearn.neighbors import KNeighborsClassifier
# модуль, где лежат различные метрики
from sklearn.metrics import accuracy_score
```

In [24]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
knn = KNeighborsClassifier(n_neighbors=3)
```

In [25]:
```
knn.fit(train_data, train_y)
```

Out[25]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [26]:
```
predicted = knn.predict(val_data)
predicted
```

Out[26]:
```
array([0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0], dtype=int64)
```

In [27]:
```
np.array(val_y)
```

Out[27]:
```
array([0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1], dtype=int64)
```

In [28]:
```
accuracy_score(predicted, val_y)
```

Out[28]:
```
0.7350746268656716
```

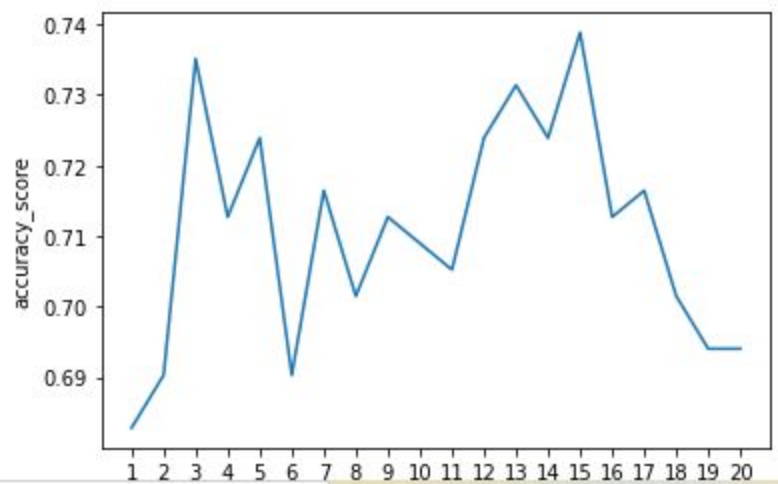
Подбор гиперпараметра k

Давайте переберем k от 1 до 20 включительно, обучим для каждого значения k классификатор KNN и посчитаем метрику accuracy_score на валидационной выборке.

In [29]:
```
val_scores = []
for k in range(1, 21):
  knn = KNeighborsClassifier(n_neighbors=k)
  knn.fit(train_data, train_y)
  predicted = knn.predict(val_data)
  acc_score = accuracy_score(predicted, val_y)

  val_scores.append(acc_score)
```

In [30]:
```
plt.plot(list(range(1, 21)), val_scores)
plt.xticks(list(range(1, 21)))
plt.xlabel("количество соседей")
plt.ylabel("accuracy_score")
plt.show()
```



Часть 3. Получение ответов для тестового датасета.

In [31]:
```
test_data = pd.read_csv("test.csv")
test_data.head()
```

Out[31]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

Предобработаем тестовый датасет **точно так же**, как тренировочный:

Предобработаем тестовый датасет **точно так же**, как тренировочный:

In [32]:
```
# удаляем колонки
test_data = test_data.drop(["PassengerId", "Name", "Ticket"], axis=1)
# заполняем пропуски
test_data["Age"] = test_data["Age"].fillna(28)
test_data["Embarked"] = test_data["Embarked"].fillna('S')
test_data = test_data.drop("Cabin", axis=1)
# переводим категориальные признаки в числовые
test_data["Sex"] = test_data["Sex"].astype('category')
test_data["Sex"] = test_data["Sex"].cat.codes

test_data = pd.get_dummies(test_data, columns=["Embarked"])
test_data.head()
```

Out[32]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S
0	3	1	34.5	0	0	7.8292	0	1	0
1	3	0	47.0	1	0	7.0000	0	0	1
2	2	1	62.0	0	0	9.6875	0	1	0
3	3	1	27.0	0	0	8.6625	0	0	1
4	3	0	22.0	1	1	12.2875	0	0	1

Посмотрим, нет ли в тестовых данных пропусков, которых не было в тренировочном датасете:

In [33]:
```
test_data.isna().any()
```

Out[33]:
```
Pclass        False
Sex           False
Age           False
SibSp         False
Parch         False
Fare           True
Embarked_C    False
Embarked_Q    False
Embarked_S    False
dtype: bool
```

Заполним пропуски в колонке Fare медианой всех значений **тренировочного датасета**:

In [34]:
```
test_data["Fare"] = test_data["Fare"].fillna(train_data["Fare"].median())
```

Обучим KNN на всей тренировочной выборке с оптимальным параметром k (полученным после подбора параметров выше):

In [35]:
```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(data, y)
```

Out[35]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [36]:
```
test_predicted = knn.predict(test_data)
```

In [37]:
```
test_predicted
```

Out[37]:
```
array([0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1],
      dtype=int64)
```

Сохраним ответы на тестовый датасет в нужном виде для отправки в соревнование:

In [38]:
```
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html
test_predicted = pd.DataFrame({"Survived": test_predicted})
test_predicted["PassengerId"] = list(range(892, 892+len(test_data)))
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html
test_predicted.to_csv("test_predicted.csv")
test_predicted
```

Out[38]:

	Survived	PassengerId
0	0	892
1	0	893
2	1	894
3	1	895
4	0	896
...	...	...
413	0	1305
414	1	1306
415	0	1307
416	0	1308
417	1	1309

418 rows × 2 columns

In [ ]:

Задание: Предсказание сердечно-сосудистых заболеваний

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.model_selection import train_test_split
```

В этом задании мы будем решать задачу предсказания наличия у человека сердечно-сосудистых заболеваний по некоторым признакам человека. Это датасет с реальными данными людей, он использовался для соревнования по машинному обучению на платформе mlbootcamp:
https://mlbootcamp.ru/round/12/sandbox/

Давайте посмотрим на датасет:

In [ ]:
```
# в этом csv файле в качестве разделителей использовалась точка с запятой,
# не забудем указать это при вызове функции read_csv
data = pd.read_csv("https://raw.githubusercontent.com/Yorko/mlcourse.ai/master/data/mlbootcamp5_train.csv", sep=';')
data.head()
```

В этом датасете данные о 70000 человек, о каждом из которых известно:

-- id человека (id) -- возраст человека в днях (age) -- пол (gender) -- рост в сантиметрах (height) -- вес в килограммах (weight) -- верхнее артериальное давление (ap_hi) -- нижнее артериальное давление (ap_lo) -- показатель холестерина (cholesterol, 1, 2 или 3) -- показатель глюкозы (gluc) -- курит ли человек (smoke, 0--не курит, 1--курит) -- употребляет ли человек алкоголь (alco, 0--нет, 1--да) -- ведет ли активную жизнь (active)

Целевая переменная: cardio -- наличие у человека сердечно-сосудистого заболевания. 1 -- есть, 0 -- нет.

In [ ]:
```
data.describe()
```

Часть 1. Предобработка датасета.

В этом датасете нет пропусков, так что заполнять NaN не требуется.

1. Разделим данные и целевую переменную cardio.

Задание:

Поедлите датасет data на данные (data) и целевую переменную (y):

In [ ]:
```
y = <тут ваш код>
data = <тут ваш код>
```

In [ ]:
```
# не меняйте код в этой ячейке!
# эта ячейка проверяет ваш код на правильность
# если при запуске ячейка выдает ошибку, то у вас в коде ошибка

assert y.shape == (70000,)
assert set(data.columns) == set(['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo',
       'cholesterol', 'gluc', 'smoke', 'alco', 'active'])
```

2. Выкинем ненужные столбцы.

В этом датасете, кажется, все признаки информативны, кроме одного -- id. Как и в Titanik, здесь id -- это просто последовательные числа, присвоенные людям при составлении датасета. Никакой информации они не несут. Давайте выкинем столбец id:

Задание:

Удалите из data колонку id

In [ ]:
```
# оставьте в переменной data все столбцы, кроме id
data = <тут ваш код>
```

In [ ]:
```
# не меняйте код в этой ячейке!
# эта ячейка проверяет ваш код на правильность
# если при запуске ячейка выдает ошибку, то у вас в коде ошибка

assert set(data.columns) == set(['age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo',
       'cholesterol', 'gluc', 'smoke', 'alco', 'active'])
```

3. Посмотрим на то, какого типа колонки в датасете.

In [ ]:
```
print(data.dtypes)
```

Кажется, что все они числовые и категориальных признаков нет. Но давайте присмотримся: признак cholesterol содержит три вида значения: 1, 2 и 3. Они выражают три уровня холестерина.

С одной стороны, между этими тремя уровнями есть связь: 3 > 2 > 1. С другой стороны, этот признак можно интерпретировать как категориальный и поделить его на три столбца.

Как будет лучше для решения задачи, мы не знаем. Нужно пробовать разные варианты, чем и занимаются data scientists. Давайте создадим копию наших данных, в которых поделим колонку cholesterol на три колонки:

Задание:

Поделите колонку cholesterol на три колонки с помощью pd.get_dummies() и запишите полученный датасет в новую переменную data_ch:

In [ ]:
```
data_ch = <тут ваш код>
```

In [ ]:
```
# не меняйте код в этой ячейке!
# эта ячейка проверяет ваш код на правильность
# если при запуске ячейка выдает ошибку, то у вас в коде ошибка

assert set(data_ch.columns) == set(['age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'gluc', 'smoke',
       'alco', 'active', 'cholesterol_1', 'cholesterol_2', 'cholesterol_3'])
```

4. Колонка Age

Если у разных признаков в датасете разные разбросы значений. функция расстояния KNN -- евклидова, то один признак будет влиять на расстояние между объектами больше, чем другой. У нас в датасете есть признак Age, который измеряется десятками тысяч, в то время как остальные признаки измеряются в единицах или десятках. Давайте сделаем копии датасетов data и data_cholesterol, в которых переведем возраст людей из дней в годы. Перевод осуществим просто поделив колонку Age на 365.

Задание:

Для копий data_age и data_ch_age датасетов data и data_ch переведите их колонки age из дней в годы, поделив колонки на 365:

In [ ]:
```
data_age = data.copy()
data_ch_age = data_ch.copy()

data_age["age"] = <тут ваш код>
data_ch_age["age"] = <тут ваш код>
```

In [ ]:
```
# не меняйте код в этой ячейке!
# эта ячейка проверяет ваш код на правильность
# если при запуске ячейка выдает ошибку, то у вас в коде ошибка

assert 64 < data_age["age"].max() < 65
assert 30 > data_age["age"].min() > 29
assert 54 > data_age["age"].mean() > 53
```

Отлично, теперь у нас есть четыре датасета:

data -- изначальный датасет без колонки id

data_ch -- изначальный датасет без колонки id и обработанным признаком cholesterol

data_age -- изначальный датасет без колонки id и колонкой age, переведенной из дней в года

data_ch_age -- изначальный датасет без колонки id и обработанным признаком cholesterol и колонкой age, переведенной из дней в года

Осталось разбить все данные на train и val. Давайте оставим под val 30% датасета:

In [ ]:
```
data_train, data_val, y_train, y_val = train_test_split(data, y, test_size=0.3)
data_ch_train, data_ch_val, y_ch_train, y_ch_val = train_test_split(data_ch, y, test_size=0.3)
data_age_train, data_age_val, y_age_train, y_age_val = train_test_split(data_age, y, test_size=0.3)
data_ch_age_train, data_ch_age_val, y_ch_age_train, y_ch_age_val = train_test_split(data_ch_age, y, test_size=0.3)
```

Часть 2. Обучение KNN

В качестве метрики качества для нашей задачи мы возьмем не accuracy, а log_loss: http://wiki.fast.ai/index.php/Log_Loss

log_loss -- это метрика для бинарной классификации (для многоклассовой ее использовать нельзя). Она принимает на вход реальные метки класса в виде нулей и единиц и вероятности принадлежности элементов первому классу, выданные алгоритмом.

В sklearn метрика log_loss определена в sklearn.metrics (как и accuracy и многие другие метрики). Чем log_loss ниже, тем ответы алгоритма "лучше" (в отличие от accuracy).

Задание:

Обучите KNN для k=3 на каждом из четырех полученных датасетов и проверьте каждую обученную модель на тесте для метрики log_loss. На каком датасете KNN с k=3 показал себя лучше всего?

Обратите внимание, что для метрики log_loss ответы KNN должны быть в виде вероятностей принадлежности классу 1 (полученными с помощью .predict_proba), а не бинарными!

In [ ]:
```
# обучаем KNN на тренировочной части data_train датасета data
knn = <тут ваш код определения knn с _neighbors=3 на data_train>
<тут ваш код обучения knn>

# получаем ответы KNN на валидационной части data_val датасета data

# predict_proba выдает массив пар -- для каждого объекта соответствующая пара
# это вероятности его принадлежности к классу 0 и классу 1.
# Чтобы получить только вероятности принадлежности классу 1,
# нужно взять 1 столбец полученного питру array: array[:, 1]
predicted_proba = knn.predict_proba(data_val)[:, 1])
log_loss(y_val, predicted_proba)
```

In [ ]:
```
<повторите то же самое для остальных трех датасетов. Не забудтье кроме train_data менять также data_val в валидации.>
```

Мы видим, что во всех четырех случаях log_loss получился довольно большим. Давайте подберем гиперпараметр k, чтобы алгоритм лучше работал:

Часть 3. Подбор параметра k

Задание:

Для датасетов data_train и data_ch_age_train обучите KNN на тренировочных выборках и найдите значения log_loss на валидационных выборках для значений k от 1 до 20 и постройте график зависимости log_loss от k (как сделано выше). Свои выводы опишите ниже.

In [ ]:
```
<тут ваш код для датасета data_train>
```

In [ ]:
```
<тут ваш код для датасета data_ch_age_train>
```

Сделайте выводы -- как вы думаете, почему зависимость log_loss от k получилась такой?

<ваши выводы опишите здесь>

Спасибо за внимание!