Елена Капаца

Машинное обучение доступным языком

Елена Капаца

Машинное обучение доступным языком

http://www.litres.ru/pages/biblio_book/?art=69273385
SelfPub; 2023
ISBN 978-5-0060-1962-1

Аннотация

Краткий гайд для новичков по машинному и глубокому обучению с разбором кода. Здесь вы найдете необходимый минимум по предмету, истолкованный языком, понятным школьнику. Некоторые разделы написаны с помощью chatGPT. По прочтении вы избавитесь от страха перед технологией и освоите базовый инструментарий подготовки данных, их загрузке в модель и ее донастройки. Подходит студентам технических специальностей.

Содержание

Елена Капаца
Машинное обучение доступным языком

Введение

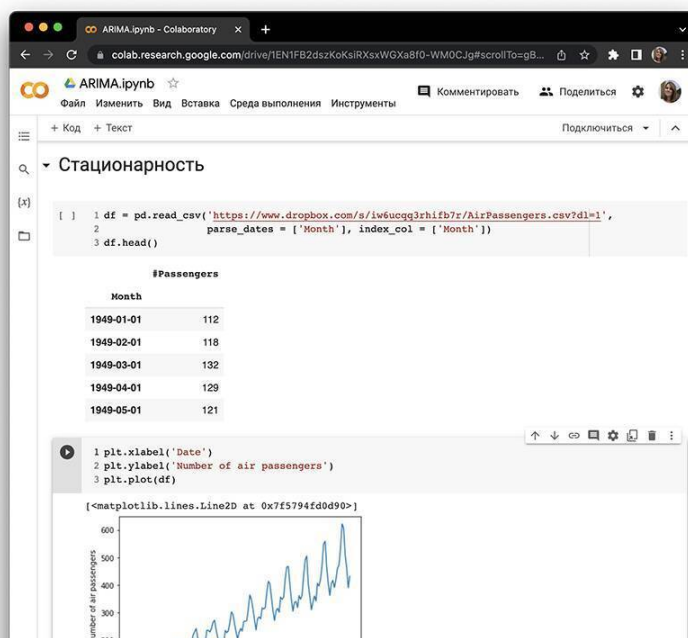
Приступая к изучению машинного обучения, студенты легко и непринужденно добиваются… запутанности. Пара-тройка непонятных терминов или неясностей при расчете – и все: мозг теряет нить и начинает воспринимать “по диагонали”. Продираться через непонятное довольно трудоёмко. Не каждый день у нас есть ресурс догугливать непонятное. Не каждый запрос в Google даст лаконичный понятный ответ.

Моя задача – описать детали этой мозаики языком, понятным старшекласснику. Я намеренно буду избегать формул, потому что знаю: каждая из них сокращает число читателей. Однако в книге будет код, и он будет расширенно комментироваться.

Минимальное требование к читателю – знание основ Python. Книга фокусируется на машинном обучении, и потому останавливаться на терминах вроде “переменной” и “списка” я не буду.

Если вы чувствуете, что пересиливаете себя при чтении, лучше сделайте перерыв. В Data Science будет предостаточно информации, однако в этой книге я постаралась собрать повторяющиеся в работе термины. Добиться их понимания особенно важно.

Некоторые главы будут базироваться на полноценных моделях и скриптах. В машинном обучении принято использовать так называемые ноутбуки – наборы ячеек с исполняемыми кусками кода:



Все используемые в дальнейшем ноутбуки можно открыть, запустить и скопировать себе для дальнейших экспериментов. Инструменты ML имеют свойство совершенствоваться, а это значит, что спустя 3-4 года после выхода книги некоторые участки кода вам придется отлаживать с помо-

щью поисковиков.

Машинное обучение – это абстрактная концепция. Ее основные компоненты стоит описывать просто, пускай даже это вызовет раздражение профессионалов. Эта книга – серия взаимосвязанных статей. Их основная цель – осветить основные и популярные термины во взаимосвязи друг с другом. Ключевые понятия при первом упоминании я буду дополнять англоязычным термином. Так вы всегда сможете с легкостью отыскать дополнительные материалы.

Немалое влияние на меня оказал бестселлер Максима Ильяхова и Людмилы Сарычевой "Пиши, сокращай". Потому эта книга написана в информационном стиле[1] и изобилует упрощениями. Если вы сохраните по прочтении ощущение удобства чтения и желание взбираться на эту познавательную гору дальше, то моя цель достигнута.

Вы всегда можете "напитаться" полноценными зубодробительными статьями на моем сайте helenkapatsa.ru.

Приятного чтения! Я буду рада вашим предложениям и фидбэку в целом (kapatsahelen@gmail.com). Вы также можете запросить PDF-версию с цветовой разметкой кода. Это упростит восприятие материала.

---

[1] Информационный стиль – инструмент для очистки текста от лишнего, для обнажения самой сердцевины текста.

Машинное обучение

Что же это такое? Машинное обучение (machine learning, ML) – наука о том, как заставить компьютеры выполнять объемную вычислительную задачу без явного программирования.

Классическим алгоритмам дают точные и полные правила для выполнения задачи, моделям Машинного обучения – данные. Мы говорим, что «подгоняем модель к данным» или «модель обучена на данных».

Проиллюстрируем это на простом примере. Предположим, мы хотим спрогнозировать цену дачного дома на основе:

- площади
- размера придомового участка
- количества комнат.

Мы могли бы попытаться построить классический алгоритм, который решает эту проблему. Этот алгоритм возьмет три вышеупомянутых признака (feature) и выдаст прогнозируемую цену на основе явного правила. Но на практике эта формула часто неочевидна.

Однако мы хотим автоматизировать этот процесс и построить модель. Она будет корректировать формулу сама каждый раз, когда появляются новые примеры цен на жилье. В целом, ML невероятно полезно для задач, когда мы

располагаем неполной или слишком обильной информацией для программирования вручную. В этих случаях мы можем предоставить имеющиеся сведения и позволить ей «изучить» недостающую. Затем алгоритм будет использовать статистические методы для извлечения недостающих знаний.

Машинное обучение способно выполнять широкий спектр задач:

- оценки стоимости чего угодно
- изменение изображений
- помощь на письме
- обработка звука
- генерация текста и многие другие.

Представьте, что Машинное обучение – это конвейер по сборке автомобилей. И первое, что потребуется для его работы – металл, различные композитные материалы, и в конечном итоге, топливо. Вся эта троица олицетворяет данные.

Данные

Данные – основа основ в ML. В контексте науки принято рассматривать два типа: традиционные и большие (big data).

Традиционные данные структурированы и хранятся в базах, управляемых с одного компьютера. На самом деле, эпитет «традиционный» введен для ясности: это помогает подчеркнуть различия с большими.

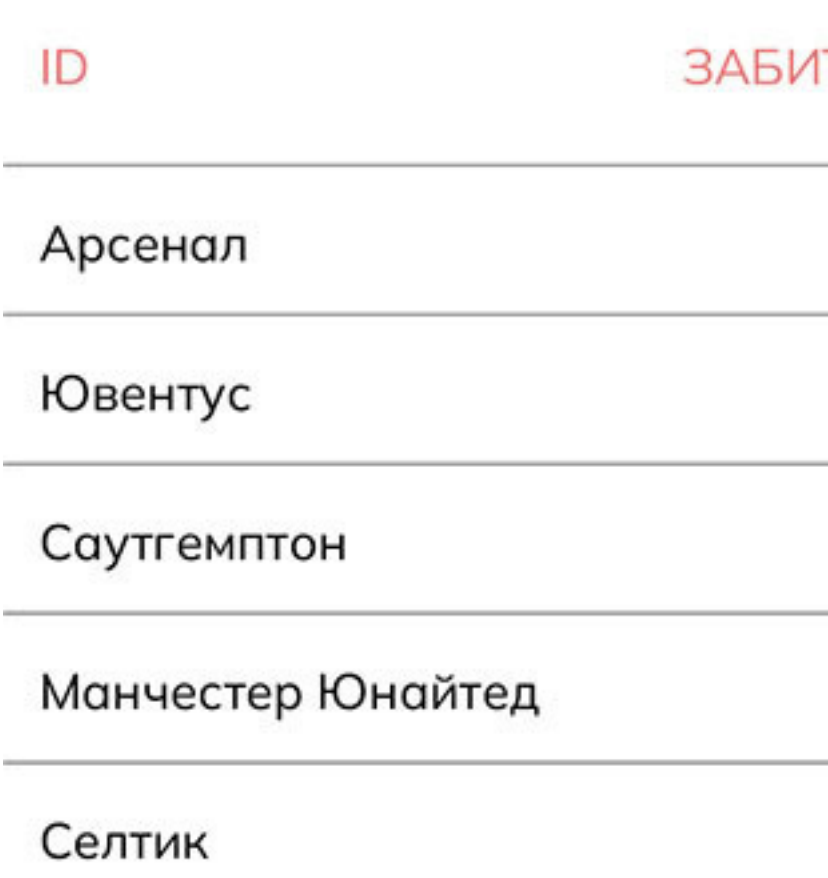
Большие данные, в свою очередь, массивнее, чем традиционные, по ряду характеристик:

- типы (числа, текст, изображения, аудио, видео и проч.)
- скорость извлечения и вычисления
- объем (тера-, пета-, эксабайты и проч.).

Набор однотипных данных, выделенный с целью обучения модели, называют датасетом (dataset). Их разделяют на следующие категории:

Классическая таблица

Здесь каждая строка имеет одинаковый набор характеристик-столбцов. Такие таблицы – датафреймы (dataframe) обычно хранятся либо в файлах форматов .csv, .parquet, либо в базах данных:



Датасет о результативности футбольных команд

Текстовый документ

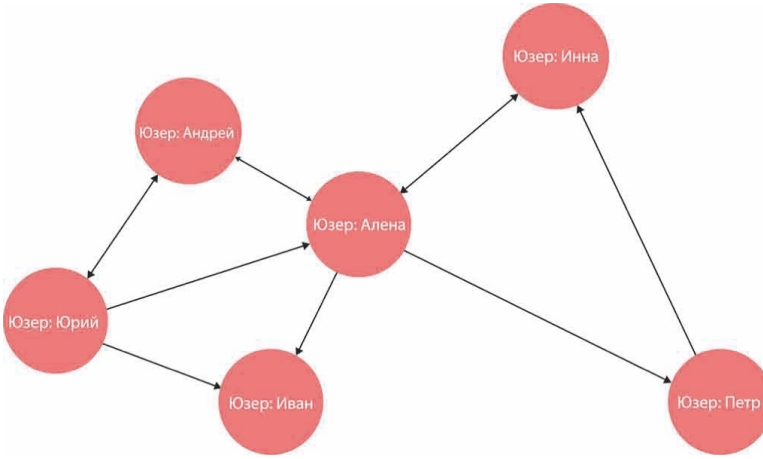
(document) Отдельно взятой единицей здесь является блок (corpus). Например, книгу можно рассматривать как датасет, состоящий из абзацев – корпусов.

"… После обучения в Университете Вашингтона Болл опубликовала статью в Journal of the American Chemical Society и отправилась на Гавайи, чтобы стать магистром химии. В 1915 г. она впервые среди женщин и афроамериканцев получила степень магистра в Гавайском колледже, где осталась преподавать".

Корпус из книги-датасета Рейчел Свайби "52 упрямые женщины"

Графы

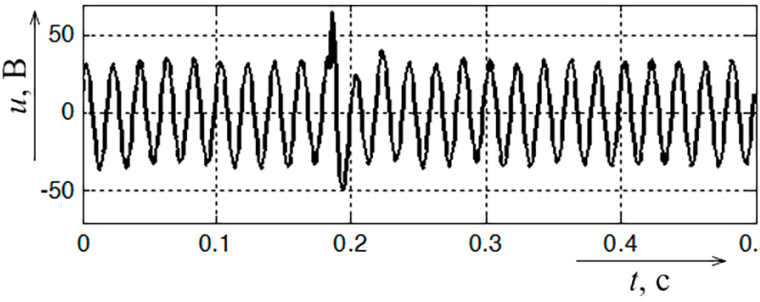
(graph) Здесь отдельно взятая единица – это связь между объектами:



Граф социальной сети

Аудиодорожки

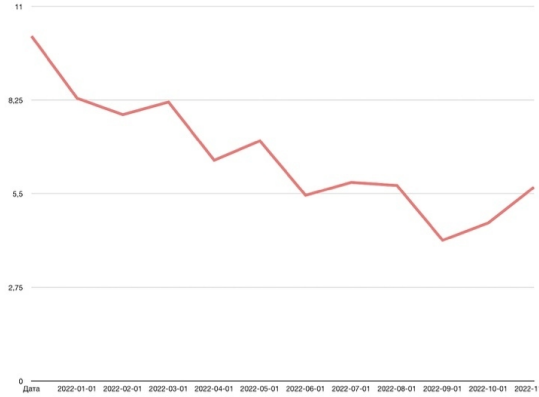
Здесь довольно очевидно: аудиозаписи. Помимо распознавания речи ML решает обширный спектр задач с помощью таких данных: очистка от шумов, написание музыки.



Временной ряд

(time series) Здесь каждая точка привязана к временной оси x и, как правило, взаимосвязана с окружающими ее соседями.

ДАТА	ЦЕНА ОТКРЫТИЯ
2022-01-01	10,130
2022-02-01	8,300
2022-03-01	7,820
2022-04-01	8,190
2022-05-01	6,480
2022-06-01	7,050
2022-07-01	5,450
2022-08-01	5,830
2022-09-01	5,740
2022-10-01	4,130
2022-11-01	4,640
2022-12-01	5,690



Цена акции LG на момент открытия биржи на протяжении года

Последовательные данные

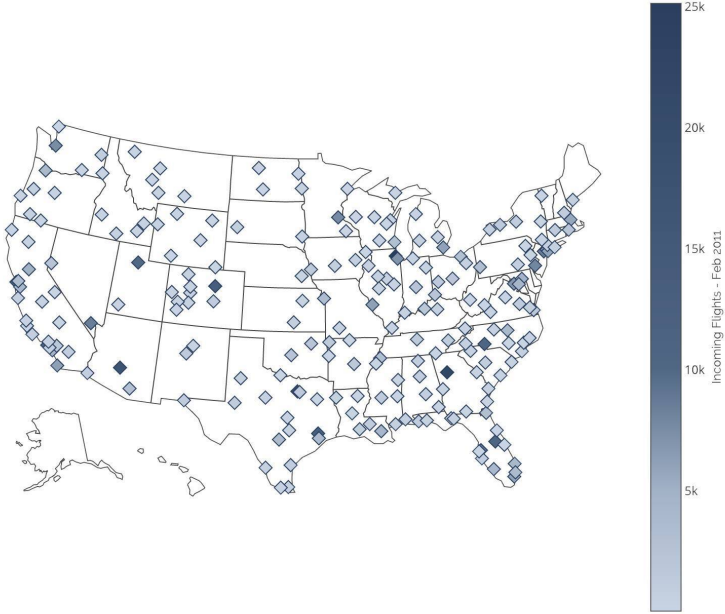
A	G	T	T	T	A	C	C
C	G	C	G	C	A	A	A
A	T	G	A	T	T	C	C
A	T	C	G	T	T	G	C
A	G	A	C	G	G	T	T
C	C	T	T	A	T	G	A
G	G	C	A	T	G	A	T
G	C	C	C	C	C	A	G

(sequence data) Состоят из набора отдельных объектов, таких как слова или буквы. Здесь нет временных меток; вместо этого есть позиции в упорядоченной последовательности:

На картинке справа яркий пример: геном – набор генов в хромосоме.

Пространственные данные

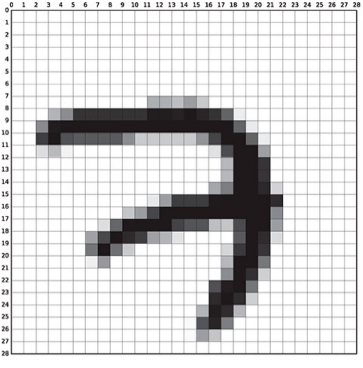
(geospatial data) Здесь каждая точка имеет координаты:



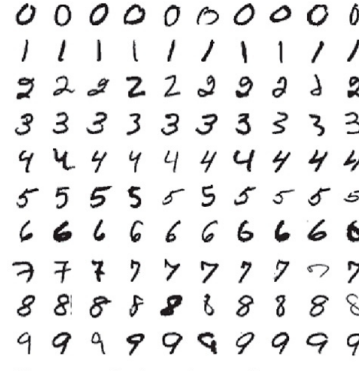
Трафик аэропортов США

Изображения

Здесь единицей является отдельная картинка. Видео рассматривается как набор картинок.



(a) MNIST sample belonging to the digit '7'.



(b) 100 samples from the MNIST training set.

Датасет рукописных цифр

Перед дата-сайентистами часто встает вопрос: где взять данные?

Студентам проще: у некоторых обширных библиотек вроде Scikit-learn встречаются собственные встроенные датасеты, прекрасно подходящие для обучения:

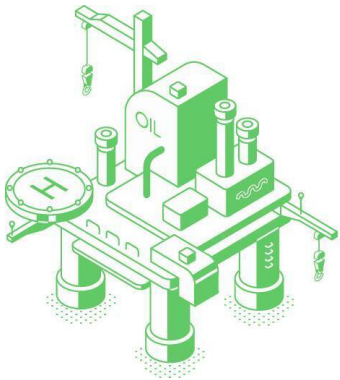
```
from sklearn.datasets import load_digits
digits = load_digits()
```

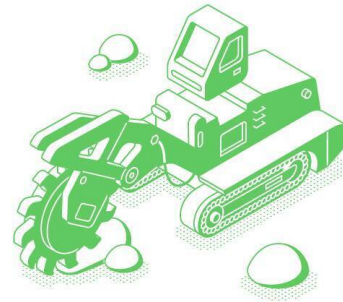
Помимо таких встроенных коллекций, данные предостав-

ляют бесплатно еще и ресурсы вроде kaggle.com.

А вот на рабочей ниве требования к информации куда специфичнее. Порой проще и лучше собрать свой набор, и в таком случае мы обращаемся к инструментам ETL.

ETL





(extract, transform, load – извлечь, преобразовать и загрузить) группа процессов, происходящих при переносе данных из нескольких систем в одно хранилище.

Если у вас есть данные из нескольких источников, вам необходимо:

• Извлекать данные из исходного источника

• Преобразовывать информацию путем очистки, объединения и других способов подготовки

• Загружать результат в целевое хранилище

Как правило, один инструмент ETL выполняет все три шага. Пожалуй, самый популярный сегодня представитель

такого программного обеспечения – это Hadoop.

ETL уходит своими корнями в 1970-е годы к появлению централизованных хранилищ данных. Но только в конце 1980-х и начале 1990-х годов, когда они заняли центральное место, мир ощутил потребность в специализированных загрузочных инструментах. Первым пользователям нужен был способ извлекать информацию из разрозненных систем, преобразовывать ее в целевой формат и загружать в конечное место хранения. Первые инструменты ETL были примитивными, и объем данных, которые они обрабатывали, был скромным по сегодняшним меркам.

По мере роста объема данных росли и хранилища данных, а программные инструменты ETL множились и становились все более сложными. Но до конца 20-го века хранение и преобразование данных осуществлялось в основном в локальных хранилищах. Однако произошло нечто, навсегда изменившее наш взгляд на хранение и обработку.

Облачные вычисления

Объем данных, которые мы генерируем и собираем, продолжает расти с экспоненциальной скоростью. У нас есть все более сложные инструменты, которые позволяют нам использовать все наши данные для получения представления о исследуемом предмете в режиме онлайн.

Традиционная инфраструктура не может масштабиро-

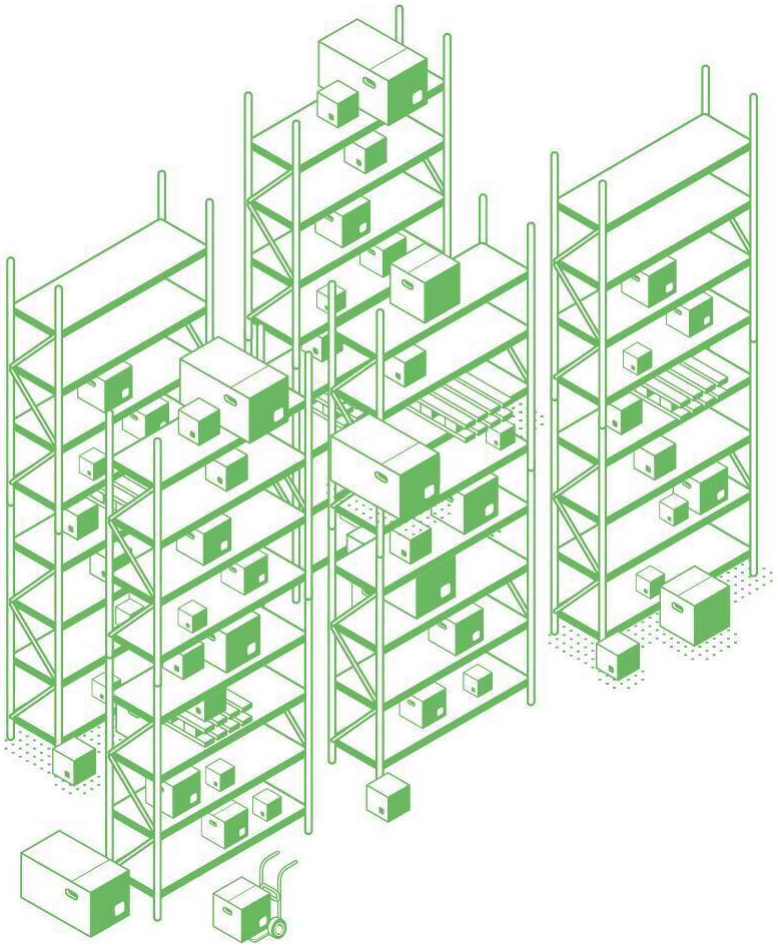
ваться для хранения и обработки большого объема данных. Это неэффективно с точки зрения затрат. Если мы хотим выполнять высокоскоростную, сложную аналитику и строить подобные модели, облако – оптимальное решение.

Облачные хранилища могут бесконечно масштабироваться для размещения практически любого объема данных. Облачное хранилище также позволяет координировать огромные рабочие нагрузки между группами вычисляющих серверов.

Преобразования и моделирование данных часто выполняются с помощью SQL – языка запросов к базе данных.

Конечная точка ETL – хранилище данных (DWH).

DWH



(data warehouse – хранилище данных) предназначено ис-

ключительно для выполнения запросов и часто содержит большие объемы исторических данных. Данные в хранилище обычно поступают из широкого круга источников, таких как:

- Логи приложений
- Сведения, собираемые с форм на сайте
- Записи различных устройств, вроде видеокамер и датчиков температуры

Хранилище объединяет большие объемы данных из нескольких источников. Это позволяет генерировать ценные инсайты[2] и улучшать процесс принятия решений. С ростом объема и качества DWH становится бесценным объектом для бизнес-аналитики. Типичное хранилище данных часто включает следующие элементы:

- Реляционная база данных
- ПО для ETL
- Инструменты анализа и визуализации
- Модели машинного обучения

К популярным хранилищам можно отнести Amazon Redshift, Google BigQuery и Greenplum.

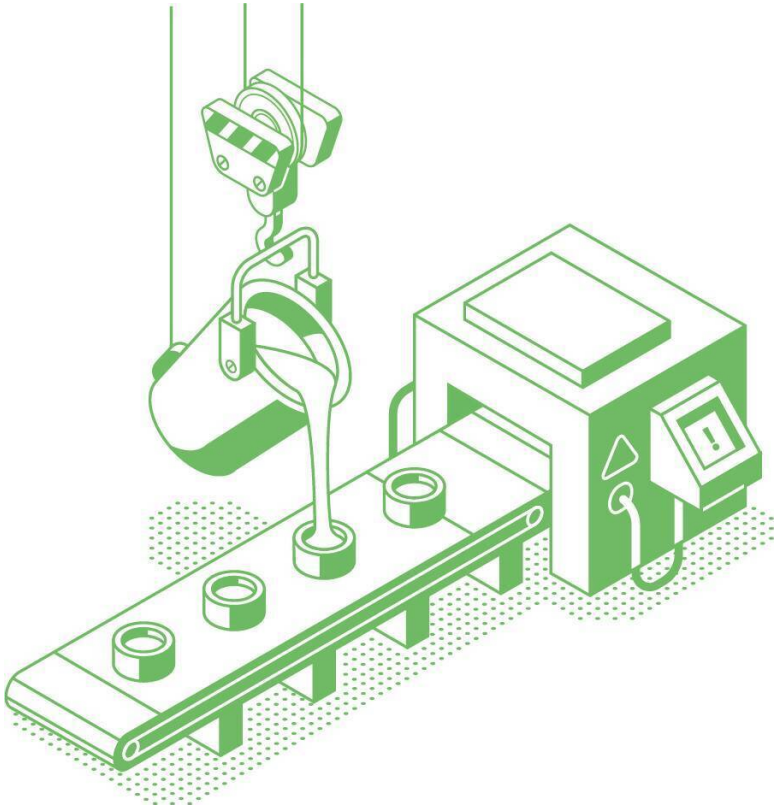
Стоит отличать DWH от так называемого озера данных (data lake). Хранилище содержит очищенные и структурированные данные, готовые к анализу на основе предопределенных потребностей бизнеса. В озере же все содержится в необработанном, неструктурированном виде.

---

[2] Решение задачи

Когда команда ML получает доступ к такому хранилищу, то предваряет моделирование целой серией действий – разведочным анализом данных (EDA).

EDA



(exploratory data analysis – разведочный анализ данных)

предварительное исследование датасета с целью определения его основных характеристик, взаимосвязей между признаками, а также сужения набора методов, используемых для создания модели.

Давайте рассмотрим, на какие этапы разбивают EDA. Для этого мы используем данные[3] банка, который продает кредитные продукты своим клиентам. Возьмет ли клиент кредит или нет?

Мы располагаем обширным набором переменных (столбцов):

[3] Исходная англоязычная версия датасета: kaggle.com/datasets/volodymyrgavrysh/bank-marketing-campaigns-dataset

ПРИЗНАК	КРАТКОЕ ОПИСАНИЕ	ТИП ДАННЫХ	ПРИМЕР
Возраст		Числовой	
Работа	Профессия	Категориальный	Менеджер, рабочий, предприниматель, домработница
Семейный статус		Категориальный	в разводе, женат / замужем, не женат / не замужем, неизвестно
Образование		Категориальный	Базовое (4 класса), неграмотный, университетское образование
Кредитный дефолт	Было ли невыполнение договора займа?	Булевый	Нет, да, неизвестно
Ипотека	Есть ли ипотека?	Булевый	Нет, да, неизвестно
Займ	Есть ли займ?	Булевый	Нет, да, неизвестно
Контакт	Контактный вид связи	Категориальный	Сотовый телефон, городской телефон
Месяц	Последний контактный месяц года	Категориальный	Янв, фев, мар, ..., ноя, дек
День недели	Последний контактный день недели	Категориальный	Пн, вт, ..., сб, вс
Длительность	Продолжительность последнего звонка в секундах	Числовой	
Кампания	Количество контактов, выполненных во время этой кампании	Числовой	
День	Количество дней, прошедших с момента последнего обращения	Числовой	
Предыдущий контакт	Количество контактов, выполненных до этой кампании	Числовой	
Доходность	Результат предыдущей маркетинговой кампании	Категориальный	Отсутствует, присутствует, неизвестно
Колебание уровня безработицы	Отклонение от базового коэффициента занятости	Числовой	
Индекс потребительских цен	Измерение среднего уровня изменения цен на товары и услуги за определённый период в экономике	Числовой	
Индекс потребительской уверенности	Степень оптимизма относительно состояния экономики	Числовой	
Европейская межбанковская ставка предложения на три месяца	Усреднённая процентная ставка по межбанковским кредитам	Числовой	
Количество сотрудников в компании	Количество работников	Числовой	
Y	Взял ли клиент кредитный продукт	Булевый	

Это не сам датасет, а только описание столбцов

Столбец Y назван так неслучайно: это общепринятое обозначение целевой переменной (target variable). Изучив 40 тысяч записей о клиентах, модель автоматически сможет предсказывать, возьмет новый клиент кредит или не возьмет.

Довольно увесистый датасет: записей в нем более 40 тысяч. Для начала[4] импортируем датасет и посмотрим на "шапку". С помощью метода head() мы отобразим шапку датафрейма и первые пять записей:

```
df = pd.read_csv('https://www.dropbox.com/s/62xm9ymoaunnfg6/bank-full.csv?dl=1', sep=';')
df.head()
```

Параметр sep используется, чтобы задать нестандартный разделитель данных по столбцам, в данном случае – точку с запятой.

[4] Здесь и далее ячейка с импортом библиотек будет пропущена. С полной версией кода можно ознакомиться в конце главы по QR-коду со ссылкой.

№	ВОЗРАСТ	РАБОТА	СЕМЕЙНЫЙ СТАТУС	...	ЕВРОПЕЙСКАЯ МЕЖБАНКОВСКАЯ СТАВКА	КОЛИЧЕСТВО СОТРУДНИКОВ В КОМПАНИИ	Y
1	27	Самозанятый	Не женат / не замужем	...	5,045	5195,8	Нет
2	30	Предприниматель	Женат / замужем	...	5,045	5195,8	Да
3	39	Голубой воротничок	Женат / замужем	...	5,045	5195,8	Да
4	42	Менеджер	Женат / замужем	...	5,045	5195,8	Да
5	42	Самозанятый	Женат / замужем	...	5,045	5195,8	Да

Все столбцы мы отображать здесь, конечно, не будем

Удаление дубликатов

(duplicates removing) Повторяющиеся записи искажают статистические показатели. Всего несколько повторов – и среднее значение столбца сместится в их пользу. Дубликаты также снижают качество обучения модели. Для начала уточним, сколько у нас строк с помощью df.shape. Затем удалим повторы с помощью drop_duplicates() и обновим данные о размере данных:

```
print(df.shape)
df.drop_duplicates(inplace=True)
print(df.shape)
```

Библиотека pandas вообще сопровождает любителей и профессионалов на каждом шагу, так что у некоторых ее компонентов параметры одинаковые. Чтобы удалить повторы "на месте", без излишнего перекопирования датафрейма, дополняем drop_duplicates() параметром inplace, равным True.

Ячейка выдает, что удалила 41188 – 41176 = 12 дубликатов:

```
(41188, 21)
(41176, 21)
```

Хоть число и небольшое, все же качество набора мы повысили.

Обработка пропусков

(omission handling) Если пропусков у признака-столбца слишком много (более 70%), такой признак удаляют. Проверим, насколько разрежены наши признаки:

```
df.isnull().mean() * 100
```

Метод isnull() пройдется по каждой ячейке каждого столбца и определит, кто пуст, а кто нет. Метод mean() определит концентрацию пропусков в каждом столбце. На 100 мы умножаем, чтобы получить значение в процентах:

```
Возраст 0.000000
Работа 0.801438
Семейный статус 0.194288
Образование 4.201477
Кредитный дефолт 0.000000
Ипотека 0.000000
Займ 0.000000
Контакт 0.000000
Месяц 0.000000
День недели 0.000000
Длительность 0.000000
Кампания 0.000000
День 96.320672
Предыдущий контакт 0.000000
Доходность 0.000000
```

Колебание уровня безработицы 0.000000
Индекс потребительских цен 0.000000
Индекс потребительской уверенности 0.000000
Европейская межбанковская ставка 0.000000
Количество сотрудников в компании 0.000000
у 0.000000

Среди всех признаков слишком много пропусков оказалось у фактора “День” (более 96%). Он подлежит удалению с помощью drop():

```
df = df.drop(columns=['День'])
```

Разберемся для начала с категориальными переменными, объединив их в один вектор. Метод pandas.unique() выделит уникальные значения из всего перечня столбцов column_values. pandas принято сокращать псевдонимом до pd.

```
column_values = df[['Работа', 'Семейный статус', 'Образование', 'Контакт', 'Месяц', 'День недели', 'Доходность']].values.ravel()
unique_values = pd.unique(column_values)
print(unique_values)
```

Список получится совсем уж нелогичный, но это здесь не столь важно. Мы лишь ищем способы обозначения пропусков. Они обозначаются словом "Неизвестно":

['Самозанятый', 'Не женат / не замужем', 'Университетская степень', 'Городской телефон', 'Октябрь', 'Пятница', 'Отсутствует', 'Предприниматель', 'Женат / замужем', 'Голу-

бой воротничок', 'Базовое (9 классов)', 'Менеджер',

'Высшая школа', 'Базовое (4 класса)', 'Техник', 'Профессиональный курс', 'Разведен(-а)', 'Неизвестно', 'Сотовый телефон', 'Август', 'Понедельник', 'Студент', 'Домохозяйка', 'Обслуживающий персонал', 'Базовое (6 классов)',

'Пенсионер', 'Четверг', 'Вторник', 'Не присутствует', 'Июль', 'Среда', 'Июнь',

'Неграмотный', 'Май', 'Ноябрь', 'Присутствует', 'Самозанятый', 'Декабрь', 'Март', 'Апрель', 'Сентябрь']

Процесс обработки пропусков можно сократить с помощью sklearn.impute.SimpleImputer. Мы выбираем все категориальные переменные и применяем стратегию "вставить вместо пропуска самое распространенное значение":

imputer = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')

categorical_columns = ["Работа", "Семейный статус", "Образование", "Месяц", "День недели", "Доходность"]

df[categorical_columns] = imputer.fit_transform(df[categorical_columns].values)

Признаки, принадлежащие к булевому типу данных, обрабатываются алгоритмом тем же образом:

imputer = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')

boolean_columns = ["Кредитный дефолт", "Ипотека", "Займ"]

df[boolean_columns] =

imputer.fit_transform(df[boolean_columns].values)

Целевую переменную y мы не обрабатываем (если в этом столбце есть пропуски, их стоит удалить).

Подобным образом заполняются пустоты в числовых переменных, только стратегия теперь – "вставить среднее значение":

imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')

numeric_columns = ["Возраст", "Длительность", "Кампания", 'Предыдущий контакт', "Колебание уровня безработицы", "Индекс потребительских цен", "Индекс потребительской уверенности", "Европейская межбанковская ставка", "Количество сотрудников в компании"]

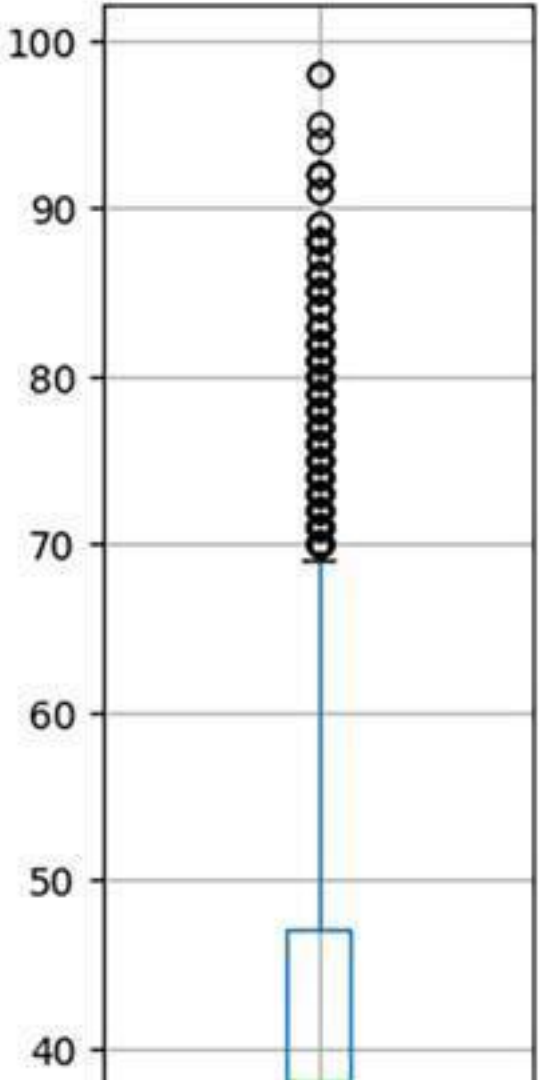
df[numeric_columns] = imputer.fit_transform(df[numeric_columns].values)

Обнаружение аномалий

(anomaly detection) Мир не идеален, и в данных бывают ошибки. Некоторые из них мы можем массово исправить с помощью доменных знаний (domain knowledge), то есть специфических сведений об исследуемой области. В мире банкинга люди старше 70 с кредитами – аномалия[5]! Их доля не столь велика, и такие записи все же вычищают.

Самый легкий способ обнаружить выбросы (outlier) – визуальный. Мы построим разновидность графика "ящик с усами" (boxplot) для одной из числовых переменных "Возраст":

[5] Сейчас картина постепенно меняется: у пожилых людей в России, например, в среднем более высокий кредитный рейтинг, и потому некоторые банки целиком таргетируют свои кредитные продукты на пенсионеров.


100
90
80
70
60
50
40

Скучковавшиеся шарики-точки в верхней части изображения – и есть аномалии. От них, как правило, избавляются с помощью квантили 99%. Это означает, что мы соберем все возрасты в виртуальный список, сортируем от малого к большому, а затем уберем 1% самых больших значений.

```
q = df["Возраст"].quantile(0.99)
df[df["Возраст"] < q]
```

Аналогичным способом определяем, что для числовых переменных пропуски обозначаются числом 999 или пустой ячейкой.

Одномерный анализ

(univariate analysis)

Прежде чем применять те или иные методы обучения, нам необходимо удостовериться, что они вообще применимы к текущему датасету. Хоть данные и ценны, к ним все равно есть требования. С этим нам поможет замечательная библиотека pandas-profiling.

Описательная статистика

Запустим профайлер и передадим наш датафрейм в качестве аргумента:

profile = ProfileReport(df)

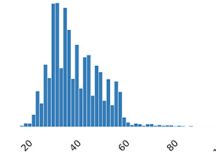
profile

Профайлер высчитает основные статистические метрики для каждой переменной и датасета в целом:

Возраст

Real number (ℝ)

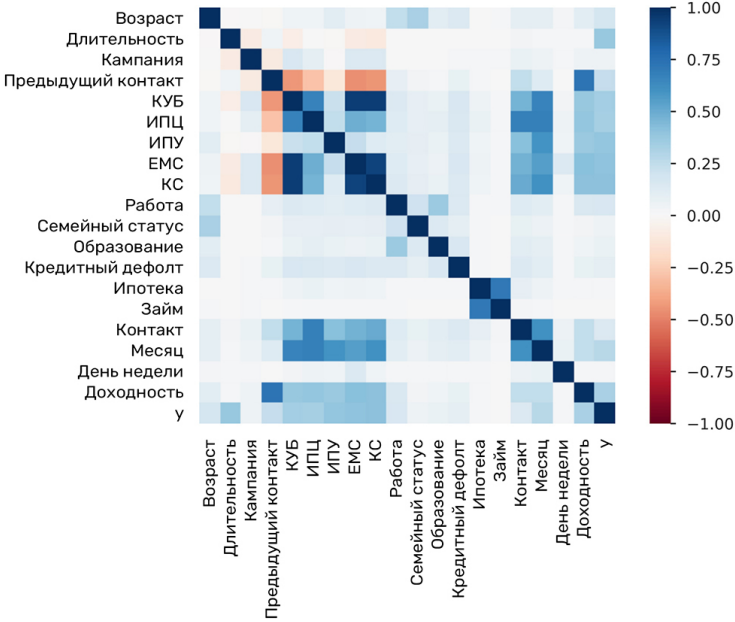
Distinct	78	**Minimum**	17
Distinct (%)	0.2%	**Maximum**	98
Missing	0	**Zeros**	0
Missing (%)	0.0%	**Zeros (%)**	0.0%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	40.0238	**Memory size**	643.4 KiB



Основные характеристики фактора "Возраст" – число уникальных значений (Distinct), пропуски (Missing), число сильно отстоящих значений (Infinite) – способ обнаружения пропусков, среднее арифметическое (Mean), минимум (Minimum), максимум (Maximum), число нулей (Zeros), занимаемая память (Memory size).

График справа помогает оценить нормальное распределение (normal distribution). Без нее множество статистических методов к признаку просто не применить, и некоторые модели не построить. Но к счастью, не везде.

Отдельный интерес для нас представляет раздел корреляций ('Correlations'). Чем ярче (краснее / синее) ячейка, тем сильнее выражена взаимосвязь между парой факторов. Диагональные ячейки игнорируются, поскольку являются результатом расчета коэффициента между переменной и ее копией.



Признаки важно именовать лаконично. Здесь пришлось ввести сокращения, чтобы pandas не обрезал слишком длинные названия:

- КУБ: Колебания уровня безработицы
- ИПЦ: Индекс потребительских цен
- ИПУ: Индекс потребительской уверенности
- ЕМС: Европейская межбанковская ставка
- КС: Количество сотрудников [в компании]

"Колебание уровня безработицы" и "Европейская межбанковская ставка" сильно коррелируют друг с другом, но являются второстепенными факторами. В дальнейшем их можно объединить на этапе инжиниринга признаков (feature engineering). Еще чаще корреляцию из датасета вообще устраняют. Группа сильно коррелирующих признаков не принесет много дополнительной информации, но усложнит алгоритм и повысит риск ошибки.

Важность признаков

(feature importance)

Прежде чем произвести инжиниринг признаков и сократить объем входных данных, стоит определить, какие признаки имеют первостепенную значимость, и в этом нам помогут scikit-learn и критерий Хи-квадрат (Chi-squared test). Метрика сравнивает размер любых расхождений между ожидаемыми и фактическими результатами, учитывая размер выборки и количество переменных. Он работает только с числовыми переменными. Мы закладываем такие в переменную X. В y с помощью слайсинга отправляем целевой признак.

Scikit-learn экономит время и использует одинаковые по названию методы[6]. Мы обучаем экземпляр класса

[6] К примеру, обучение модели и определение важности признаков оба производятся методами fit(), predict().

SelectKBest (“выбрать K самых важных признаков”) с помощью fit() и не ограничиваем число факторов. Для построения рейтинга фичей мы “загоняем” очки важности в датафрейм dfScores, а названия столбцов – в dfColumns.

```
X = df[['Возраст', 'Длительность', 'Кампания', 'День', 'Предыдущий контакт', 'Индекс потребительских цен', 'Европейская межбанковская ставка', 'Количество сотрудников в компании']]
y = df.iloc[:, -1]
bestFeatures = SelectKBest(score_func = chi2, k = 'all')
fit = bestFeatures.fit(X, y)
dfScores = pd.DataFrame(fit.scores_)
dfColumns = pd.DataFrame(X.columns)
```

Посмотрим на 10 самых важных признаков:

```
featureScores = pd.concat([dfColumns, dfScores], axis = 1)
featureScores.columns = ['Фича', 'Очков важности']
print(featureScores.nlargest(10, 'Очков важности'))
```

Неожиданно, но в топе оказалась “кампания”, то есть число звонков в рамках текущей рекламной кампании. Похоже, мы недооценивали силу упорства!

№	ФИЧА	ОЧКОВ ВАЖНОСТИ
1	Длительность	1.760568e+06
2	Количество сотрудников в компании	5.232760e+03
3	Европейская межбанковская ставка	3.239336e+03
4	Предыдущий контакт	3.089714e+03
5	Кампания	5.419261e+02
6	Возраст	1.031094e+02
7	День недели	2.749463e+00
8	Индекс потребительских цен	2.732241e+00

Многомерный анализ

(multivariate analysis) Чего только не создаст комьюнити в науке о данных!

Парные особенности

Для нужд разведочного анализа крайне кстати будут парные графики. Здесь на помощь приходит другой великолепный класс – seaborn.pairplot(). Каждая из переменных ляжет в основу одной из осей двумерного точечного графика. Изменим еще раз названия длинных переменных, чтобы уместить их на скромном отведенном пространстве.

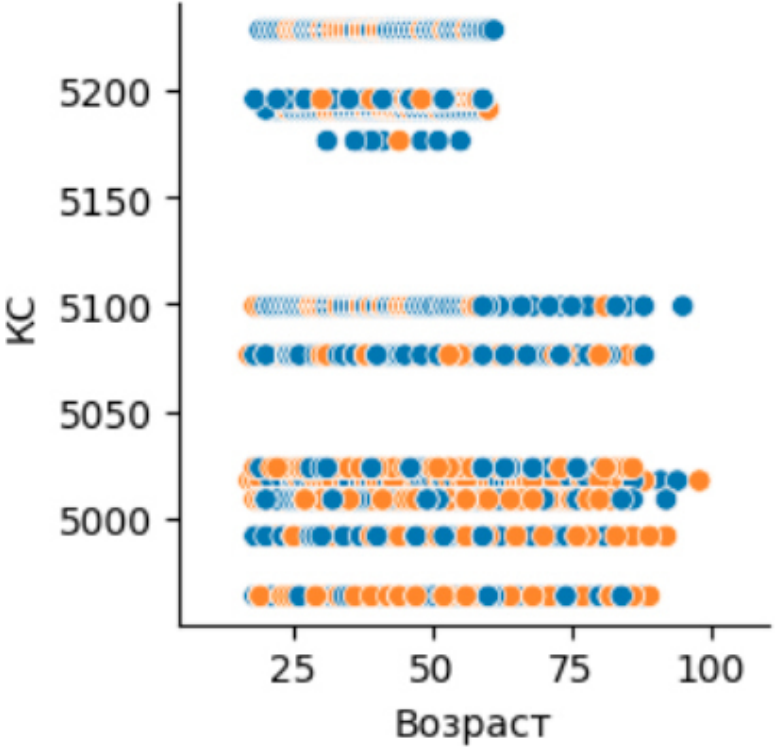
```
df.rename(columns = {
'Предыдущий контакт': 'ПК',
'Колебание уровня безработицы': 'КУБ',
'Индекс потребительских цен': 'ИПЦ',
'Индекс потребительской уверенности': 'ИПУ',
'Европейская межбанковская ставка': 'ЕМС',
'Количество сотрудников в компании': 'КС'
}, inplace = True)
sns.pairplot(df, hue = 'y')
```

Полную версию парного графика со всеми признаками, можно найти в ноутбуке.

Весь огромный график во имя читаемости здесь пока-

зывать не буду. Кратко пройдемся по некоторым находкам. Оранжевым обозначены клиенты, взявшие кредит:

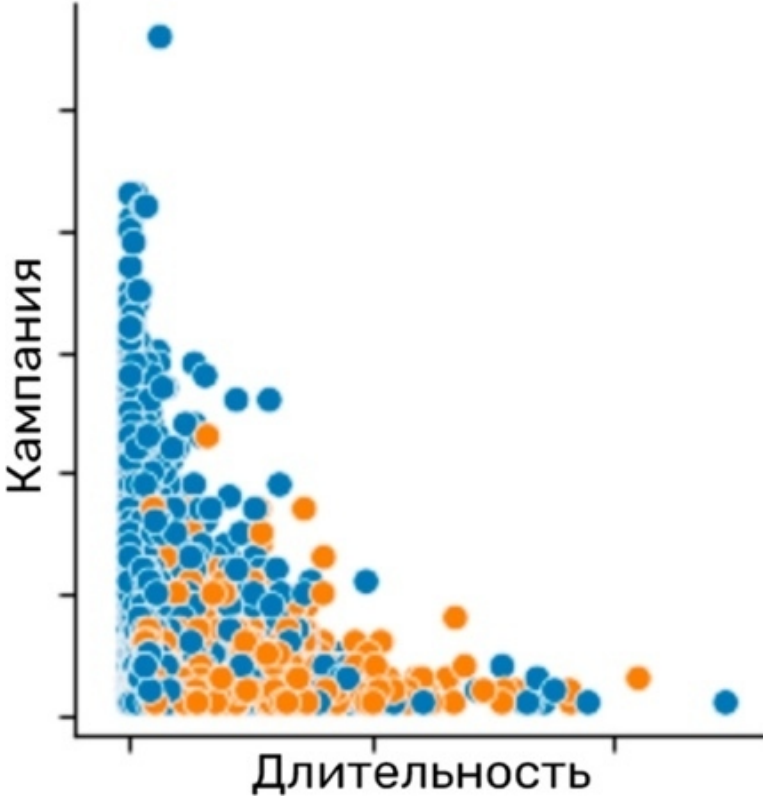
При сопоставлении возраста и количества сотрудников в компании данные быстро собрались в группы-линии:



Это означает, что вторая фича скорее относится к катего-

риальным признакам. Ее значения – ограниченный ряд чисел: 5200, 5175, 5100, 5075 и так далее.

Мы сопоставили кампании (то есть число звонков в рамках текущей рекламной кампании) и длительности звонка:



Синяя горка не взявших продукт у основания оси x говорит о том, что не готовые взять кредит люди склонны заканчивать разговор быстро. Купившие же, наоборот, любят обсудить детали. Появляется гипотеза, которую стоит проверить: если увлечь клиента разговором, шанс выдать кредит повышается.

Понижение размерности

(dimensionality reduction) Некоторые из признаков могут быть “слиты” в один с помощью специальной техники – Анализа главных компонент (PCA – principal component analysis). Давайте создадим с его помощью такой агрегирующий столбец. Он в равной мере представит исходные признаки.

Выделим список признаков, подлежащих “сокращению”. Среди них целевого быть не должно.

```
features = ['Колебание уровня безработицы', 'Индекс потребительских цен', 'Индекс потребительской уверенности', 'Европейская межбанковская ставка']
x = df.loc[:, features].values
y = df.loc[:,['y']].values
```

Преобразование данных

Помимо скраббинга и исследования данных, некоторые признаки целиком трансформируют. Это делается и для высвобождения вычислительных мощностей, и для выравнивания влиятельности признаков на предсказание.

Стандартизация

Чтобы числа разной величины в разных признаках не путали модель, приведем их к единому масштабу. В этом нам поможет стандартизация (standardization). Это преобразование значений таким образом, что из каждого наблюдения вычитается среднее значение фичи. Затем результат делится на стандартное отклонение (standard deviation) этого признака. Стандартное отклонение показывает, насколько исходное значение ячейки отклоняется от среднего значения столбца. К примеру, в столбце "Длительность" средним значением является 258 секунд. Для ряда № 4 длительность равна 103. Стандартное отклонение равно 258 – 103 = 155 секунд. Стандартизованное значение будет равно: (103 – 258) ÷ 155 ≈ 0,98.

```
x = StandardScaler().fit_transform(x)
pd.DataFrame(data = x, columns = features).head()
```

StandardScaler() на месте заменил данные на их стандар-

тизированную версию.

	КОЛЕБАНИЕ УРОВНЯ БЕЗРАБОТИЦЫ	ИНДЕКС ПОТРЕБИТЕЛЬСКИХ ЦЕН	ИНДЕКС ПОТРЕБИТЕЛЬСКОЙ УВЕРЕННОСТИ	ЕВРОПЕЙСКАЯ МЕЖБАНКОВСКАЯ СТАВКА
0	-0.11581	0.384015	0.022227	0.820856
1	-0.11581	0.384015	0.022227	0.820856
2	-0.11581	0.384015	0.022227	0.820856
3	-0.11581	0.384015	0.022227	0.820856
4	-0.11581	0.384015	0.022227	0.820856

Так мы получаем признаки, где все значения как бы отцентрованы относительно нуля. Такое преобразование также сокращает нагрузку на компьютеры.

Нормализация

Существует еще несколько возможных стадий EDA. К этому датасету их применять уже не стоит.

Один из них – это нормализация (normalization). Это похоже на перевод в процент от максимального значения. Допустим, для того же столбца “Длительность” максимальным значением является 4918 секунд. У записи № 4, где значением длительности является 103, нормализация происходит так: $103 \div 4918 \approx 0{,}02$.

Мы прошлись лишь по EDA классических табличных

данных. Для других разновидностей информации применяются совершенно иные методы.

Пересэмплирование (oversampling)

Классов у банковского датасета всего два – “Да” (взял кредит) и “Нет” (не взял). То есть мы имеем дело с двоичной классификацией (binary classification).

Уравняем доли классов с помощью техники пересэмплирования синтетического меньшинства (SMOTE). Для этого категориальные и булевые признаки придется конвертировать в числовые (такое требование у SMOTE). Перечислим еще раз категориальные признаки в виде списка. pandas по умолчанию присваивает им общий тип – “объект”, потому сменим тип на категориальный явно. pandas.Series.cat.codes заменит текст на буквы на месте.

```
categoricalColumns = ['Работа', 'Семейный статус', 'Образование', 'Контакт', 'Месяц', 'День недели', 'Доходность']
df[categoricalColumns] = df[categoricalColumns].astype('category')
df[categoricalColumns] = df[categoricalColumns].apply(lambda x: x.cat.codes)
```

Тоже самое для булевых признаков:

```
booleanColumns = ['Кредитный дефолт', 'Ипотека', 'Займ',
```

'y']

df[booleanColumns] = df[booleanColumns].astype('category')

df[booleanColumns] = df[booleanColumns].apply(lambda x: x.cat.codes)

Посмотрим, насколько сбалансирован наш датасет. Для этого разделим его на X (все факторы) и y (целевой признак):

X = df.loc[:, df.columns != 'y']

y = df.iloc[:, -1]

Подсчитаем представителей каждого из двух классов:

counter = Counter(y)

print(counter)

Разница между классами почти девятикратная:

Counter({'Нет': 36548, 'Да': 4640})

Запускаем SMOTE:

oversample = SMOTE()

X, y = oversample.fit_resample(X, y)

И вызываем счетчик классов еще раз:

counter = Counter(y)

print(counter)

Теперь наступил баланс. Модель не будет ошибаться из-за перекоса.

Counter({'Нет': 36548, 'Да': 36548})

Другие методы разведочного анализа данных

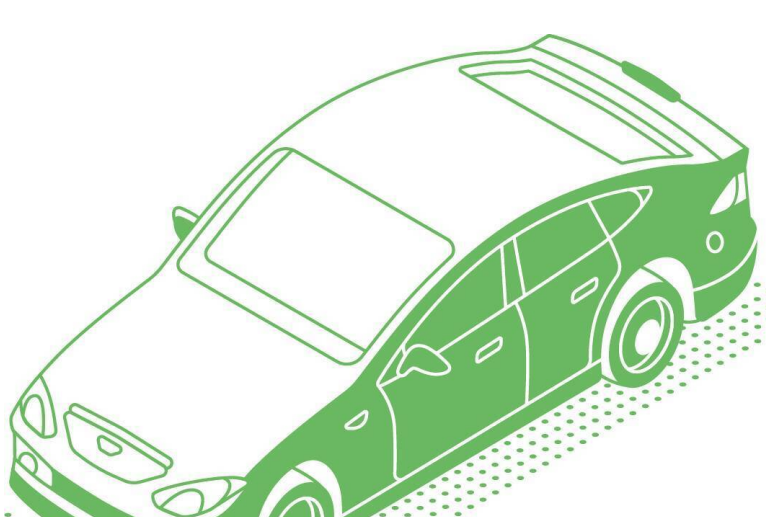
Зачастую немалый объем работы помимо EDA представляет собой скраббинг (scrubbing), в который входят:

- Исправление грамматических ошибок
- Удаление не несущих смысла частей текста (например, номер обращения из строки с заявкой пользователя)
- Удаление пустых или слишком коротких текстовых ячеек (из них не извлечь смысл)
- В отдельную категорию преобразований помещу методы, которые выполняются над текстовыми данными после EDA и в преддверии моделирования:
- Стемминг (stemming) – извлечение основ слов
- Лемматизация (lemmatization) – приведение слова к исходной форме.

Для начала более чем достаточно. Рекомендую изучать EDA других типов данных, когда наступит комфорт в обращении с классическими табличными датасетами.

Ноутбук с кодом можно пощупать с помощью QR-кода.

Модель



Модель – это код, который прогнозирует или преобразовывает. Она может выражаться одним числом, таким как среднее значение датасета, функцией y = a * x + b или набором правил дерева решений.

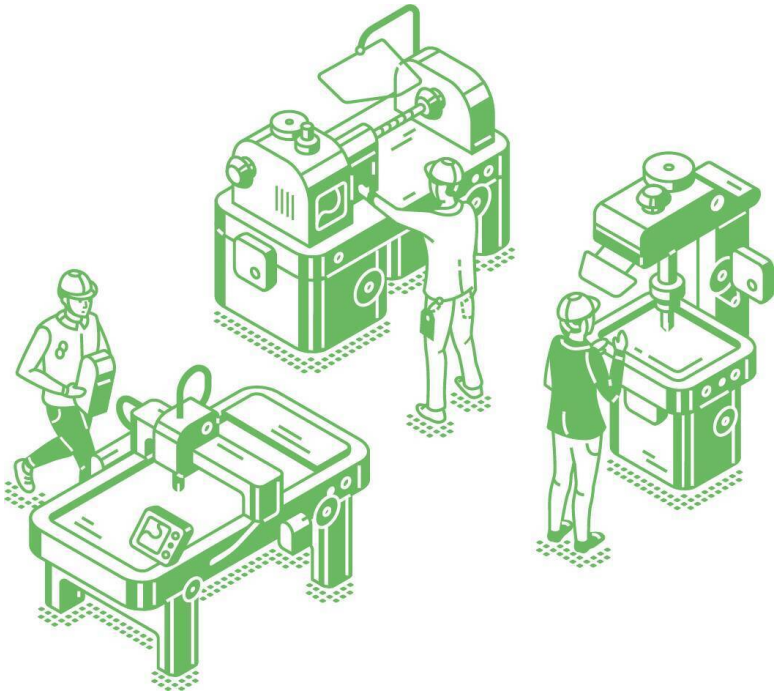
Список существующих моделей постоянно растет. Но вот

некоторые популярные из них:

- Бустинг (boosting)
- Машина градиентного бустинга (GBM)
- Дерево решений (decision tree)
- Линейная регрессия (linear regression)
- Логистическая регрессия (logistic regression)
- Метод k-ближайших соседей (kNN)
- Метод k-средних (K-Means)
- Метод опорных векторов (SVM)
- Наивный байесовский классификатор (naive Bayes)
- Случайный лес (random forest)

В отдельную категорию выносят ансамблевые модели, то есть комбинации перечисленных выше видов.

Моделирование



Давайте попробуем построить дерево решений. Этот инструмент определяет способы разделения данных на группы на основе различных условий.

Выделим столбцы-предикторы X, на которых будет основывать свое предсказание модель, и целевой признак y – ответ на вопрос “Возьмет ли клиент кредит?”:

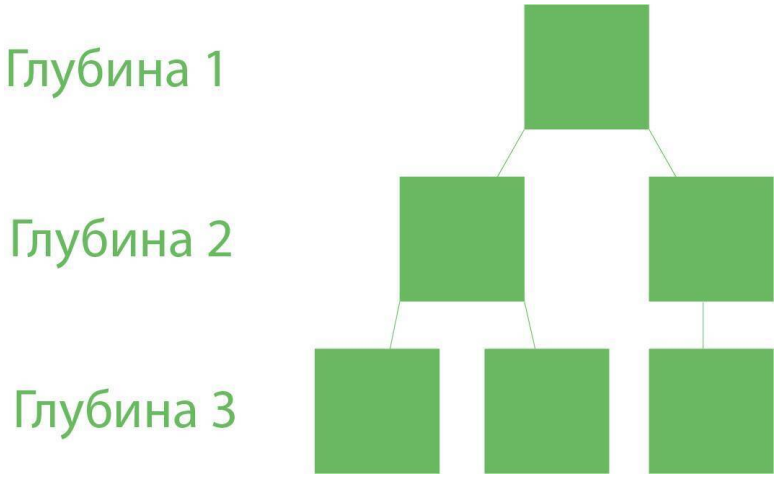
```
X = df[df.columns[:-1]]
y = df['y']
```

Разделим данные на тренировочные и тестовые части. Первые используются для обучения модели. Она изучает взаимосвязи и затем использует эти знания для прогнозирования на новых, неизвестных сведениях.

Эти новые сведения – и есть тестовые данные. Они используются для оценки производительности модели. Так мы получим представление о том, насколько хорошо модель будет работать на новой информации в будущем. test_size, равный 0.3 означает, что размер тестовой выборки составляет 30% от общего объема данных. 70% приходится, соответственно, на тренировочные.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=25, shuffle=True)

Инициируем экземпляр дерева решений. max_depth означает количество разветвлений, как бы этажность дерева:

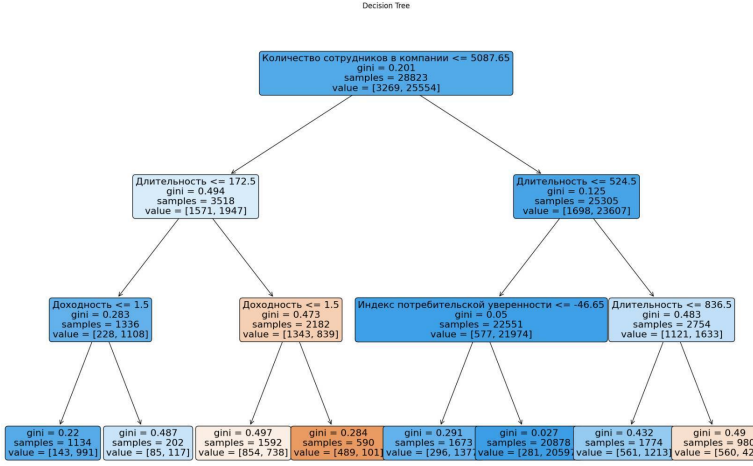


В random_state можно записать любое число. Этот параметр позволит перезапустить модель в будущем и получить подобные результаты. Метод fit() принимает тренировочную часть данных и непосредственно учит модель.

```
tr = tree.DecisionTreeClassifier(max_depth=3, random_state=25)
tr.fit(X_train, y_train)
```

Модель готова, и теперь мы визуализируем дерево с помощью matplotlib.figure(). У дерева решений scikit-learn есть свой метод plot_tree(), где лаконично указываются и названия признаков. Параметр filled, равный True, раскрашивает узлы. rounded придает узлам-прямоугольникам небольшое закругление. fontsize отвечает за размер шрифта.

```
fig=plt.figure(figsize=(23,15))
tree.plot_tree(tr.fit(X_train, y_train), feature_names=X.columns, filled=True, rounded=True, fontsize=16)
```



Часть этого дерева мы с легкостью интерпретируем сами. Если число сотрудников меньше или равно 5087, мы учитываем длительность разговора. Если длительность меньше или равна 172,5 секунд, то учитываем доходность. Если доходность меньше 1,5 (значение стандартизировано и не является суммой в валюте), 1134 не возьмут кредит, а 202 возьмут. Голубым цветом окрашен класс большинства. value по-

казывает, сколько записей классов “взял” / “не взял” попадает в категорию того или иного разветвления. Мы установили глубину дерева равной трем, так что ветвление можно продолжать.

Весьма неочевидная логика, но так в банках решения и принимают.

Вы могли подметить значения gini и samples в разветвлениях. Индекс Джини (Gini) показывает вероятность того, что случайный экземпляр будет неправильно классифицирован при случайном выборе. samples – число записей в данном разветвлении. То есть сколько из них удовлетворяют всем условиям вплоть до корня дерева (корень здесь сверху).

Пощупать модель можно с помощью QR-кода.

Настало время оценить производительность нашей модели.

Валидация

(validation – проверка) метод оценки модели. Выделяют несколько методов[7], но мы воспользуемся k-блочной кросс-

[7] С полным списком можно ознакомиться на сайте (helenkapatsa.ru/vidy-validatsii).

валидацией (k-fold cross-validation).

Чтобы свести к минимуму ошибки выборки, мы изменим формат разделения данных. Вместо того, чтобы делать одно разбиение на тренировочную и тестовую части, мы сделаем их много и проверим модель на каждой комбинации.

Преимущество заключается в том, что все наблюдения используются как для обучения, так и для проверки, а каждое наблюдение используется один раз для проверки. Обычно мы разбиваем датасет на 5 или 10 частей: это обеспечивает баланс между вычислительной сложностью и точностью.

Кросс-валидация (или перекрестная проверка) – это метод оценки модели в условиях небольшого объема данных. Данные разделяют на несколько равных подмножеств, затем запускают обучение на первом из них. Проверка качества обучения производится уже на втором подмножестве. Затем следует дообучение на втором подмножестве и оценка на третьем, и так со всеми подмножествами.

Получим список предсказаний по тестовым данным и вычислим эффективность кросс-валидации. Метод mean() усреднит показатель 10 разбиений:

```
tr_pred = tr.predict(X_test)
cv_tr = cross_val_score(tr, X_train, y_train, cv=10).mean()
```

Мы записали конечный показатель эффективности в переменную cv_tr и скоро узнаем ее значение. Этот показатель называется доля правильных ответов (accuracy). Он измеряет отношение правильно спрогнозированных наблюдений к

общему их количеству. “Сколько процентов от общего числа клиентов, взявших или не взявших кредит, модель распознала верно?”

Для оценки текущей эффективности дерева мы попробуем четыре вида метрик. Их гораздо больше, и у каждой своя специфика. Помимо точности мы посмотрим еще:

• Точность (precision), долю объектов, названных моделью положительными и при этом действительно являющимися положительными “Сколько процентов статусов клиентов модель угадала?”

• Полнота (recall): показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм. “Сколько процентов взявших кредит угадала модель?”

• F1-скор (F1 score): объединяет точность и полноту в одно число, и позволяет оценить баланс между этими двумя метриками. Это полезно, когда есть класс большинства, “сбивающий прицел” обеим метрикам выше.

```
print('Доля правильных ответов: %.3f' % tr.score(X_test, y_test))
print('Доля правильных ответов на кросс-валидации: %0.3f' % cv_tr)
print('Точность: %.3f' % precision_score(y_test, tr_pred))
print('Полнота: %.3f' % recall_score(y_test, tr_pred))
print('F1-мера: %.3f' % f1_score(y_test, tr_pred))
```

Вот такие мы получаем показатели:

Доля правильных ответов: 0.910

Доля правильных ответов во время кросс-валидации: 0.908

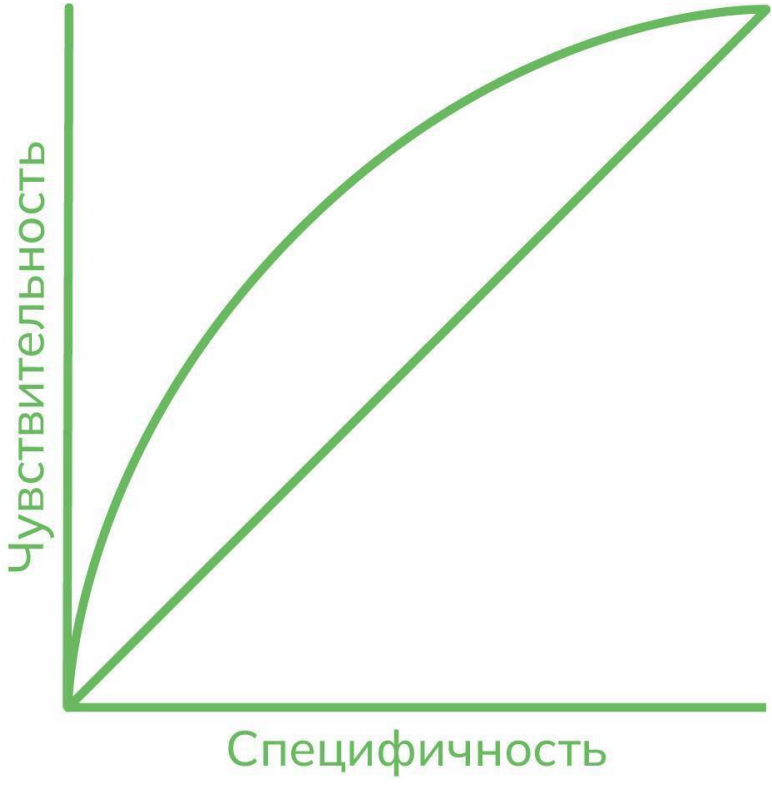
Точность результата измерений: 0.946

Отзыв: 0.953

Оценка F1: 0.950

Доля правильных ответов равна 91%. Звучит неплохо, не так ли? Внедрим для надежности еще одну метрику под названием AUC ROC (area under curve – площадь под кривой, receiver operating characteristic – рабочая характеристика приёмника). Аббревиатура расшифровывается запутанно, так что давайте разберемся постепенно.

Кривая ROC – графическое представление производительности модели:



ROC-кривая полукруглой формы

AUC ROC показывает компромисс между двумя показателями:

• чувствительность (sensitivity) – это доля наблюдений, которые являются истинно положительными (“взял кредит”) и

размечены моделью как положительные.

• специфичность (specificity) – это доля наблюдений, которые являются истинно отрицательными (“не взял кредит”) и размечены моделью как отрицательные.

Она представляет собой площадь сложной фигуры розового цвета. Значение площади принято рассчитывать таким образом, чтобы значение находилось в диапазоне от нуля до единицы.

Модель с AUC, равной единице, является идеальным классификатором. То есть розовая фигура якобы заполонила собой все пространство в левой верхней половине графика.

Прямая из начала координат под углом 45 градусов – это изображение случайного угадывания. При сбалансированных классах вероятность угадать равна 50%, то есть 0.5 на языке AUC.

Если кривая “уплощается” и как бы совпадает такой прямой, то модель работает как случайное угадывание. Такой результат никого не устроит.

AUC ROC – это способ измерить, насколько хорошо модель может различать два класса. Более высокий показатель означает лучшую производительность модели.

Scikit-learn и тут упрощает нам подсчет площади ROC-кривой:

```
tr_probs = tr.predict_proba(X_test)
tr_probs = tr_probs[:, 1]
```

```
auc_tr = roc_auc_score(y_test, tr_probs)
print('AUC: %.2f' % auc_tr)
```

Показатель близок к единице, и это замечательно!

```
AUC: 0.93
```

Еще один важный инструмент оценки качества – матрица ошибок (confusion matrix). Это показатель успешности классификации, таблица с четырьмя различными комбинациями:

• Истинно позитивное предсказание (true positive, сокр. TP). Модель предсказала положительный результат, и клиент действительно взял кредит. Доля таких предсказаний – уже известная нам чувствительность.

• Истинно негативные предсказание (true negative, TN). Модель предсказала отрицательный результат, и клиент действительно не взял кредит. Доля TN – это специфичность.

• Ошибочно позитивные предсказание (ошибка типа I, false positive, FN). Предсказан положительный результат (клиент возьмет кредит), но на самом деле этого не происходит.

• Ошибочно негативные предсказание (ошибка типа II, false negative, FN). Клиент не должен был взять кредит, но сделал это.

		РЕАЛЬНЫЕ ЗНАЧЕНИЯ	
		НЕГАТИВНЫЕ	ПОЗИТИВНЫЕ
ПРЕДСКАЗАННЫЕ ЗНАЧЕНИЯ	НЕГАТИВНЫЕ	ИСТИННО ПОЗИТИВНЫЕ	ОШИБОЧНО ПОЗИТИВНЫЕ
	ПОЗИТИВНЫЕ	ОШИБОЧНО НЕГАТИВНЫЕ	ИСТИННО НЕГАТИВНЫЕ

Посмотрим, как наше дерево решений умеет отделять зерна от плевел. Здесь снова нам поможет всемогущий scikit-learn и его класс confusion_matrix. Мы загружаем сгенерированные на тестовом наборе предсказания. sns.heatmap() создаст тепловую карту. Подпишем оси x и y.

```
tr_matrix = confusion_matrix(y_test, tr_pred)
sns.set(font_scale=1.3)
plt.subplots(figsize=(8,8))
sns.heatmap(tr_matrix, annot=True, cbar=False, cmap='twilight', linewidth=0.5, fmt="d")
plt.ylabel('Истинный класс')
plt.xlabel('Предсказанный класс')
plt.title('Матрица ошибок дерева решений');
```

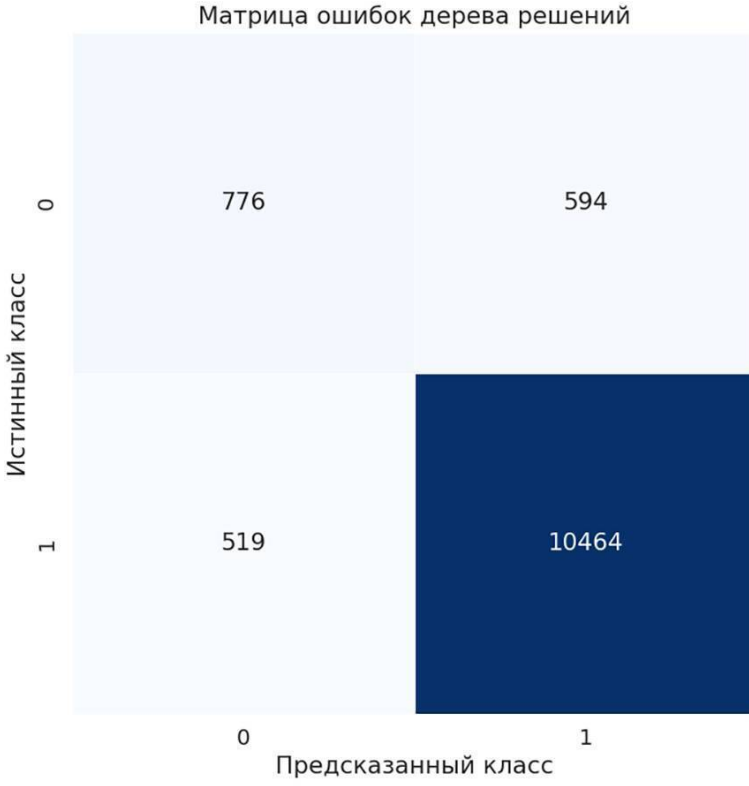
Теперь сопоставим термины с результатами:

Истинно позитивное предсказание: 10464

Истинно негативные предсказание: 776

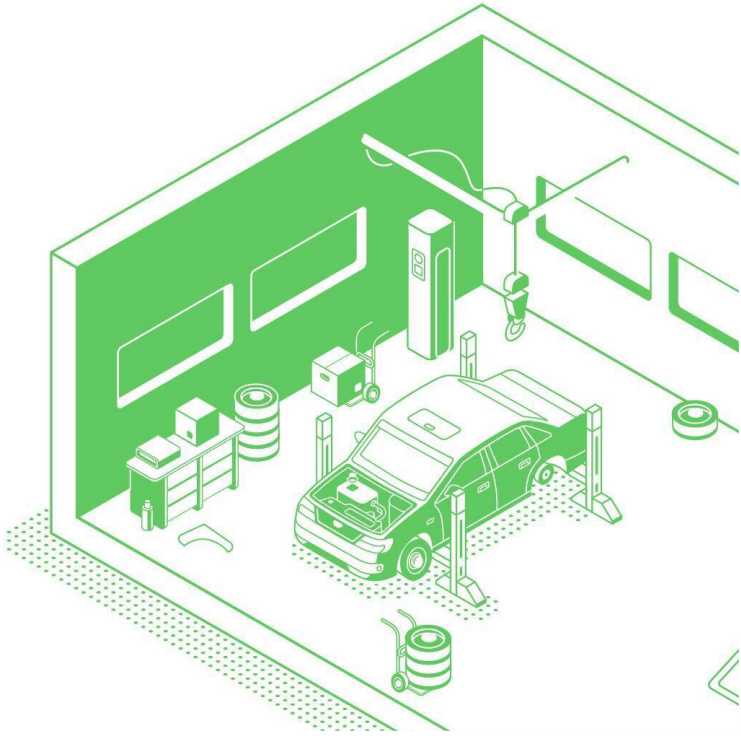
Ошибочно позитивные предсказание: 594

Ошибочно негативные предсказание: 519



Наша тестовая выборка состоит из 30% от исходного числа записей (36K). Среди них преобладает доля взявших кредит. Нам есть куда расти, модель стоит улучшить.

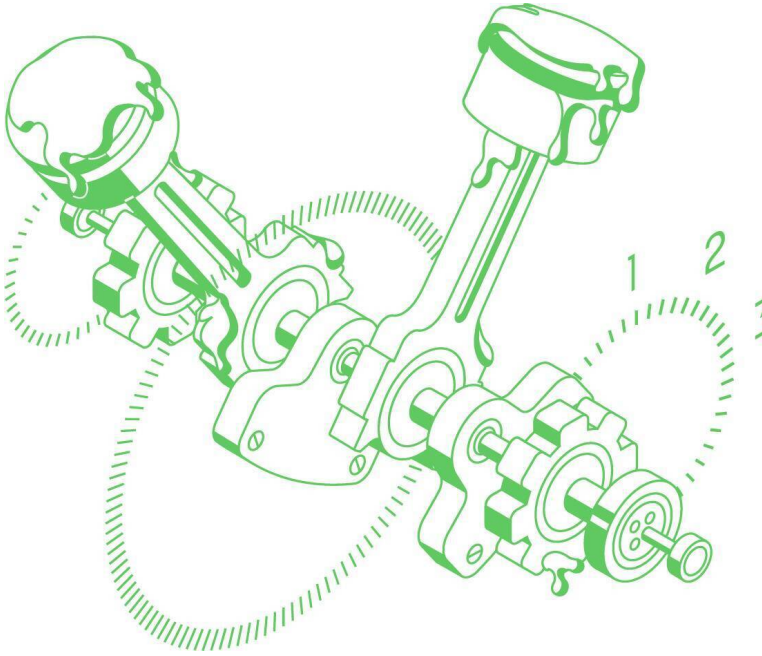
Регуляризация



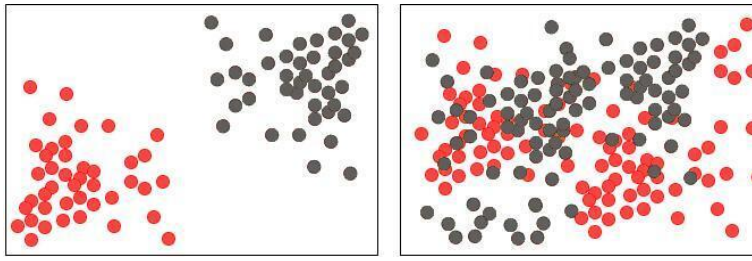
Вот тут-то на помощь и приходит регуляризация. Выражаясь простыми словами, сейчас мы будем донастраивать

модель.

Scikit-learn предлагает нам утилиту для автоматического подбора наилучших параметров дерева. Называется она GridSearchCV и позволяет использовать разные параметры регуляризации. Они аналогичны компонентам двигателя. Меняя значение одного, мы воздействуем на всю модель.



Мы возьмем индекс Gini и энтропию (entropy). Последняя означает меру примеси в наборе данных.



Низкая (слева) и высокая энтропия

Аббревиатура cv означает уже знакомую нам кросс-валидацию, то есть GridSearchCV 10 раз соберет модель, чтобы вычислить ее наилучшие параметры. n_jobs позволяет распараллелить вычисления, но для такого небольшого датасета в этом нужды нет. verbose дает посмотреть на подробные выходные сведения о процессе. max_depth позволяет сделать дерево решений высотой в 20 уровней.

```
parameters = {'criterion':['gini','entropy']
'max_depth':
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]}

default_tr = tree.DecisionTreeClassifier(random_state=25)
gs_tree = GridSearchCV(default_tr, parameters, cv=10,
n_jobs=-1, verbose=1)
gs_tree.fit(X_train, y_train)
```

Обновим значения метрик:

```
print('Лучшие параметры дерева решений: {}'.format(gs_tree.best_params_))
print('Доля правильных ответов: %0.3f' % (gs_tree.score(X_test,y_test)))
print('Доля правильных ответов кросс-валидации: %0.3f' % gs_tree.best_score_)
print('Точность: %.3f' % precision_score(y_test, gs_tree_pred))
print('Полнота: %.3f' % recall_score(y_test, gs_tree_pred))
print('F1-мера: %.3f' % f1_score(y_test, gs_tree_pred))
```

Небольшой, но все же прирост мы обеспечили. Заодно и обнаружили, что глубину стоит сделать равной пяти. Когда речь заходит о больших данных, каждые пол-процента могут означать тысячи людей и огромный прирост в денежном эквиваленте:

```
Лучшие параметры дерева решений: {'criterion': 'gini', 'max_depth': 5}
Доля правильных ответов: 0.915
Доля правильных ответов кросс-валидации: 0.913
Точность: 0.936
Полнота: 0.970
F1-мера: 0.953
```

Теперь дадим модели тестовые, неизвестные данные и соберем в gs_tree_probs ее предсказания в виде таблицы. Срез [:, 1] создаст из нее список. Оценим динамику AUC ROC.

```
gs_tree_probs = gs_tree.predict_proba(X_test)
```

gs_tree_probs = gs_tree_probs[:, 1]

gs_tree_auc = roc_auc_score(y_test, gs_tree_probs)

print('AUC: %.2f' % gs_tree_auc)

Значение площади значительно не изменилось.

AUC: 0.93

Теперь визуализируем нашу ROC-кривую со всеми прилагающимися элементами. Для этого мы создадим переменные gs_tr_fpr (доля ложно позитивных предсказаний), gs_tr_tpr (доля истинно позитивных), gs_tr_thresholds (порог отсечки).

gs_tr_fpr, gs_tr_tpr, gs_tr_thresholds = roc_curve(y_test, gs_tree_probs)

Мы дважды построим ROC-кривые: первую версию дерева решений красным цветом и улучшенную зеленым.

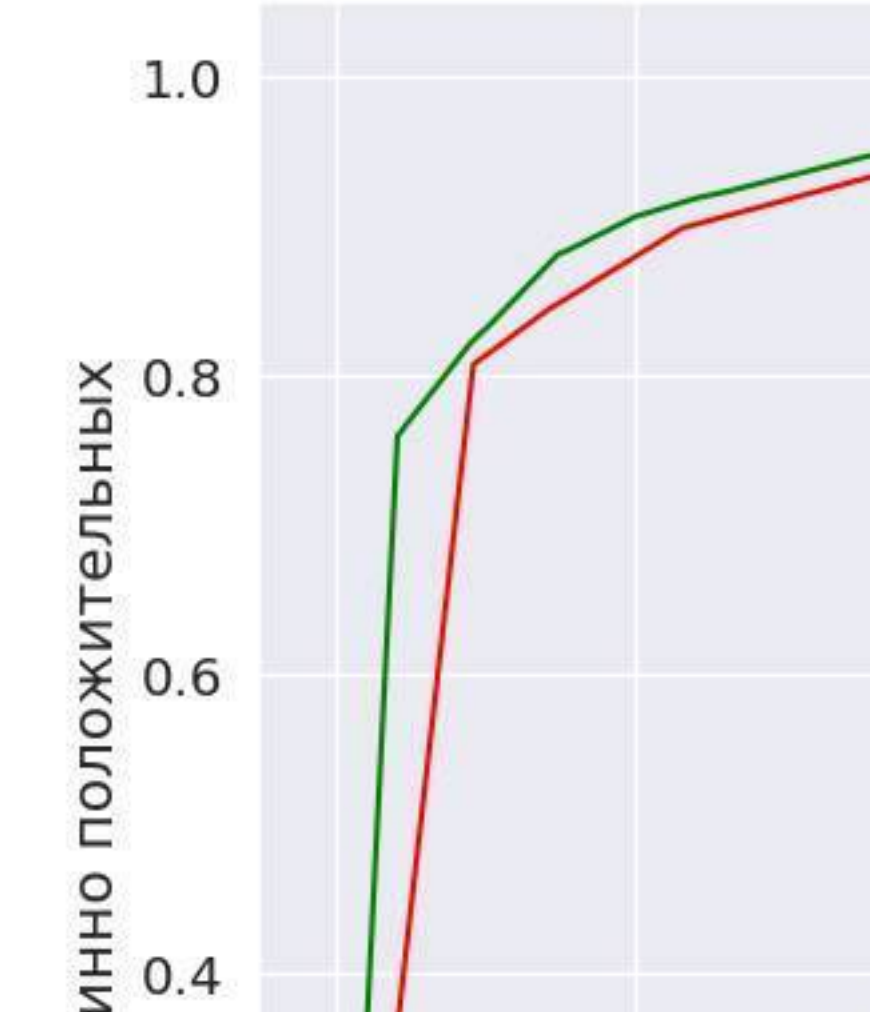
tr_fpr, tr_tpr, tr_thresholds = roc_curve(y_test, tr_probs)

Зададим размер графика в 8 x 8 дюймов. В легенду зашьем и значение площади для ROC-кривой. Построим прямую случайной классификации между двух точек с координатами [0, 1] и [1, 0]. Установим темно-синюю цветовую схему в color. Зададим заголовки осей x и y с помощью xlabel, ylabel. Прикрепим легенду к правому верхнему углу (upper left). size, равный 10, задает размер шрифта.

plt.figure(figsize=(8,8))

plt.plot(tr_fpr, tr_tpr, color='red', label='ROC-кривая дерева решения (AUC= %0.2f)'% auc_tr)

```
    plt.plot(gs_tr_fpr, gs_tr_tpr, color='green', label='ROC-кривая GridSearch + Дерево решений (AUC= %0.2f)'% gs_tr_auc)

    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='–', label = 'Случайная классификация')
    plt.xlabel('Доля ложно положительных')
    plt.ylabel('Доля истинно положительных')
    plt.title('ROC-кривая MLP')
    plt.legend(loc=”upper left”, props={‘size’: 10)
    plt.show()
```


1.0
0.8
0.6
0.4
инно положительных

Все-таки до и после подбора параметров эффективность модели разная. Даже незначительную разность площадей легко оценить визуально.

На этом закончим раздел классического моделирования в машинном обучении. Наука не стоит на месте, и в 2000-х стало популярно глубокое обучение.

Разница между машинным и глубоким обучением

Термины искусственный интеллект (artificial intelligence), машинное обучение и глубокое обучение (deep learning) даже используются взаимозаменяемо. Но каждый из них имеет свое особое значение.

В общих чертах, глубокое обучение – это подмножество машинного, а машинное – подмножество искусственного интеллекта. Их удобно представить в виде перекрывающихся концентрических кругов:

Искусственный интеллект – это теория и практика компьютерных систем, способных выполнять человеческие задачи.

Машинное обучение автоматически адаптируется к задаче с минимальным вмешательством человека.

Глубокое обучение использует искусственные нейронные сети для имитации обучения человеческого мозга.

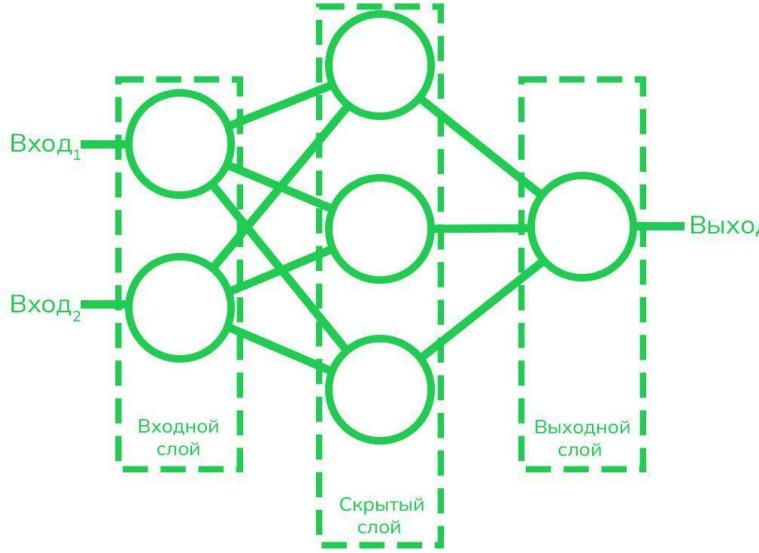
Взгляните на эти ключевые отличия между ML и DL:

МАШИННОЕ ОБУЧЕНИЕ	ГЛУБОКОЕ ОБУЧЕНИЕ
Подмножество ИИ	Подмножество машинного обучения
Может обучаться на небольших наборах данных	Требует больших объемов данных
Требует большего вмешательства человека для дообучения и исправления	Самостоятельно учиться с помощью окружающей среды и прошлых ошибок
Более короткая тренировка и больше ошибок	Более длительная тренировка и более высокая точность
Вычисляет простые паттерны	Вычисляет сложные паттерны
Может обучаться на центральном процессоре	Обучается на графическом процессоре

У нас есть прекрасная возможность опробовать настоящее глубокое обучение с помощью Google Colaboratory, ведь там предоставляется все необходимое железо. Мы попробуем освоить многослойный перцептрон.

Глубокое обучение

Ключевой концепцией DL является многослойный перцептрон (multilayer perceptron). Это разновидность искусственной нейронной сети. Она состоит из нескольких уровней взаимосвязанных узлов, каждый из которых выполняет простые вычисления:

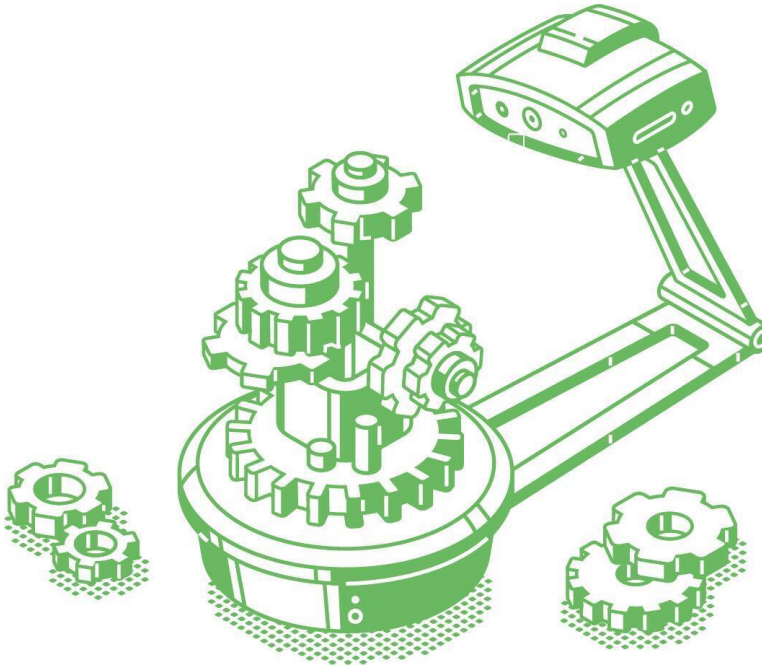


Входной слой (input layer) получает данные, которые затем передаются через один или несколько скрытых уровней.

Каждый узел в скрытом слое преобразует информацию, после чего результат передается на следующий этап.

Последний уровень – это выходной слой (output layer). Он выдает окончательный результат вычислений – метки классов, преобразованные изображения и так далее.

Если вернуться к аналогии с конвейером, то глубокое обучение – это не просто роботизированная рука, прилаживающая деталь к автомобилю. Это словно робот, способный изучать результаты своей работы и улучшаться на основании своей же деятельности.



MLP часто используются для таких задач, как распознавание изображений, обработка естественного языка и прогнозирование, когда входные данные сложны и их трудно анализировать с использованием традиционных методов программирования. Используя многослойный персептрон, алгоритм учится распознавать закономерности и делать прогнозы на основе этих закономерностей, что позволяет ему выполнять сложные задачи с высокой степенью точности.

Многослойный перцептрон

Создадим экземпляр класса MLPClassifier с сотней скрытых слоев и перемешиванием записей. Обучение будет проходить в не более чем 1000 итераций. random_state, как мы помним, может принимать любое значение. Он нужен, чтобы задавать стартовую точку при перемешивании и напрямую влияет на воспроизводимость результата в будущем. Обучим модель на тренировочных данных.

```
mlp = MLPClassifier(hidden_layer_sizes=(100), max_iter=1000, random_state=25, shuffle=True)
mlp.fit(X_train, y_train)
```

Проверим, как модель справляется. Для этого сгенерируем предсказания для тестовой части датасета. В scikit-learn возможно выполнить кросс-валидацию в одну строку кода. Поскольку у каждого этапа кросс-валидации свой показатель эффективности, то мы собираем все и усредняем значение. В следующей ячейке мы используем эту переменную.

```
mlp_pred = mlp.predict(X_test)
cv_mlp=cross_val_score(mlp, X_train, y_train, cv=10).mean()
```

В предыдущей главе дерево решений показало хороший результат: F1-мера равнялась 93,5%. Посмотрим, составит ли MLP здесь конкуренцию дереву.

```
print('Доля правильных ответов: %.3f' % mlp.score(X_test,
```

y_test))

print('Доля правильных ответов во время кросс-валидации: %0.3f' % cv_mlp)

print('Точность результата измерений: %.3f' % precision_score(y_test, mlp_pred))

print('Полнота: %.3f' % recall_score(y_test, mlp_pred))

print('Оценка F1: %.3f' % f1_score(y_test, mlp_pred))

Вполне составит! Даже в неотрегулированном виде на полпроцента обгоняет дерево решений!

Доля правильных ответов: 0.889

Доля правильных ответов во время кросс-валидации: 0.888

Точность результата измерений: 0.889

Отзыв: 1.000

Оценка F1: 0.941

Кстати, реальный мир с его многообразием данных, моделей и задач сложен. Так что единого порогового значения не существует. Однако по негласному соглашению диапазон 70% и выше считается индикатором жизнеспособности модели. Показатель выше 90% – хорошим. Модели с эффективность 95% и выше – отличные, даже превышающие возможности человека.

Аналогично дереву решений, соберем предсказания тестовой части данных и рассчитаем ROC-кривую.

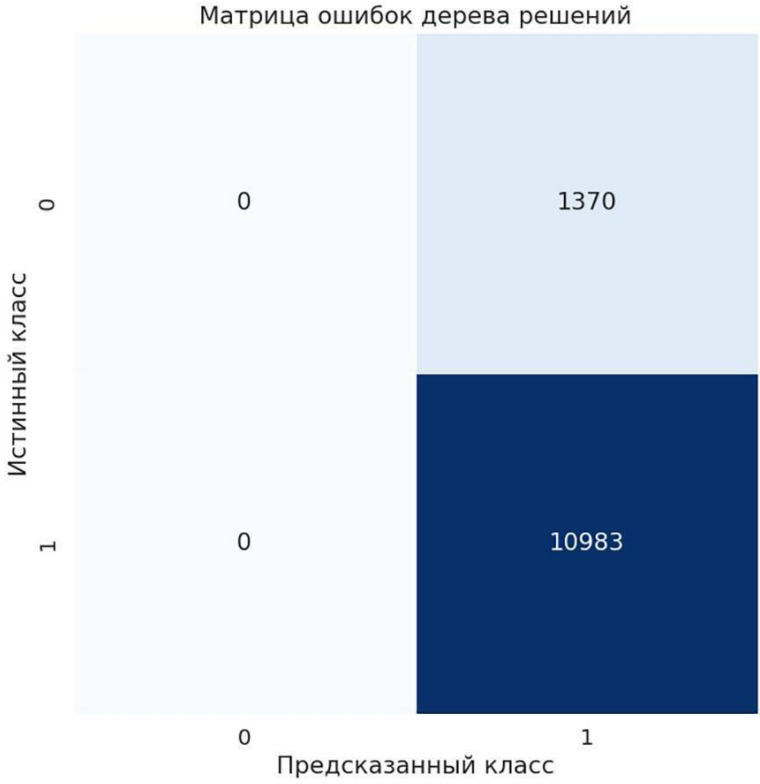
```
mlp_probs = mlp.predict_proba(X_test)
mlp_probs = mlp_probs[:, 1]
```

```
auc_mlp = roc_auc_score(y_test, mlp_probs)
print('AUC: %.2f' % auc_mlp)
```

Взглянем снова на матрицу ошибок.

```
matrix = confusion_matrix(y_test, mlp_pred)
sns.set(font_scale=1.3)
plt.subplots(figsize=(8, 8))
sns.heatmap(matrix, annot=True, cbar=False, cmap='Blues',
linewidth=0.5, fmt="d")
plt.ylabel('Истинный класс')
plt.xlabel('Предсказанный класс')
plt.title('Матрица ошибок дерева решений')
```

Значения матрицы ошибок MLP существенно отличаются от “мнения” дерева решений. MLP не поместила в нулевой класс (“не купил”) ничего. Посмотрим, хорошо это или плохо.



Следующая ячейка будет отрабатывать несколько минут. В первой ее части фигурирует сразу несколько важных терминов, и мы с ними сейчас познакомимся.

Нейроны можно исключать из процесса обучения модели, например, из-за неважности данных в нем. За это отвечает

так называемая функция активации. Не будем вдаваться глубоко в ее устройство[8]. Отмечу, что мы используем два вида – логистический (logistic) и ReLU.

Есть у модели и свой "дирижер", он же солвер (solver). Он организует оптимизацию модели, координирует обновление параметров. Мы не ограничимся единственным и используем три, чтобы потом выбрать лучший:

- LBFGS
- Adam
- SGD

Дабы сохранить читаемость углубляться в солверы не будем[9].

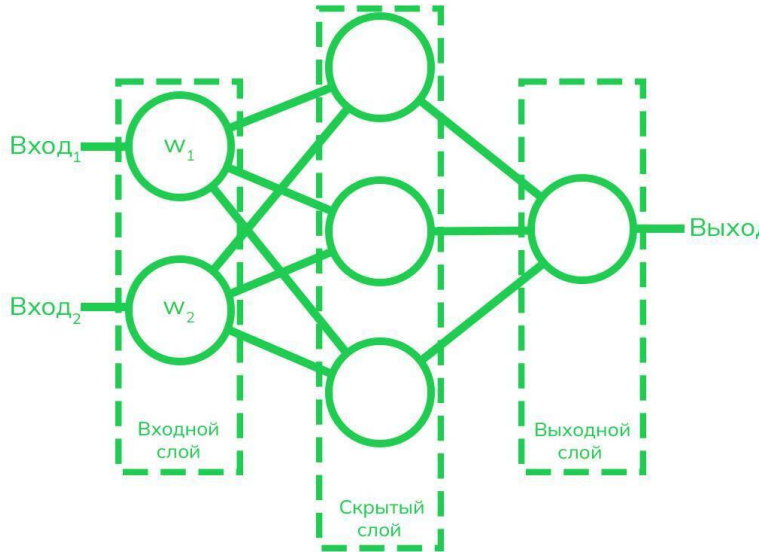
Для регуляризации MLP мы также применим GridSearchCV в 10 шагов кросс-валидации. np.arange() сгенерирует список от единицы до четырех.

Помните диаграмму многослойного перцептрона в начале главы? Дополним ее ключевым понятием DL – весами (weights). Их обозначают так: w0, w1 и так далее:

---

[8] Полный список функций активации можно найти здесь: helenkapatsa.ru/vidy-funktsii-aktivatsii.

[9] С полным списком оптимизаторов можно познакомиться подробнее в статье: helenkapatsa.ru/solver.



Это коэффициенты, которые характеризуют важность данных в том или ином нейроне. Мы уже знаем, что число сотрудников в компании является важным признаком. Значит, нейрон числа сотрудников в компании будет иметь больший вес.

Веса стоит понимать, чтобы познакомиться с Альфой (alpha) – это субалгоритм регуляризации, который изменяет веса.

Альфа-параметр в данном случае получит несколько чисел 10, возведенных в степени от -1 до -4. Мы также вручную указываем размеры скрытых слоев в hidden_layer_sizes.

parameters = {'activation':['logistic','relu'],'solver': ['lbfgs','adam','sgd'], 'alpha':10.0 ** -np.arange(1,4), 'hidden_layer_sizes':[(20),(4),(10),(14,2),(4,1),(10,5), (11,3)]}

default_mlp = MLPClassifier(random_state=25)
gs_mlp = GridSearchCV(default_mlp, parameters, cv=10, n_jobs=-1, verbose=1)
gs_mlp.fit(X_train, y_train)

Теперь можно вычислить улучшенные предсказания:

gs_mlp_pred=gs_mlp.predict(X_test)

Отобразим плоды наших трудов – наилучшие параметры MLP и обновленные метрики ее эффективности:

print("Лучшие параметры MLP: {}".format(gs_mlp.best_params_))
print('Доля правильных ответов во время кросс-валидации: %0.3f' % gs_mlp.best_score_)
print('Доля правильных ответов: %0.3f' % (gs_mlp.score(X_test,y_test)))
print('Точность результата измерений: %.3f' % precision_score(y_test, gs_mlp_pred))
print('Полнота: %.3f' % recall_score(y_test, gs_mlp_pred))
print('F1-мера: %.3f' % f1_score(y_test, gs_mlp_pred))

Регуляризация помогает:

Лучшие параметры MLP: {'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': 10, 'solver': 'lbfgs'}

Доля правильных ответов во время кросс-валидации: 0.906

Доля правильных ответов: 0.904

Точность результата измерений: 0.947

Отзыв: 0.945

Оценка F1: 0.946

Аналогично дереву решений, соберем предсказания тестовой части данных. Отобразим площадь для ROC-кривой.

```
gs_mlp_probs = gs_mlp.predict_proba(X_test)
gs_mlp_probs = gs_mlp_probs[:, 1]

gs_mlp_auc = roc_auc_score(y_test, gs_mlp_probs)
print('AUC: %.2f' % gs_mlp_auc)
```

Площадь здесь равна 0.92. Достойный результат. Скоро мы сравним кривые визуально еще раз.

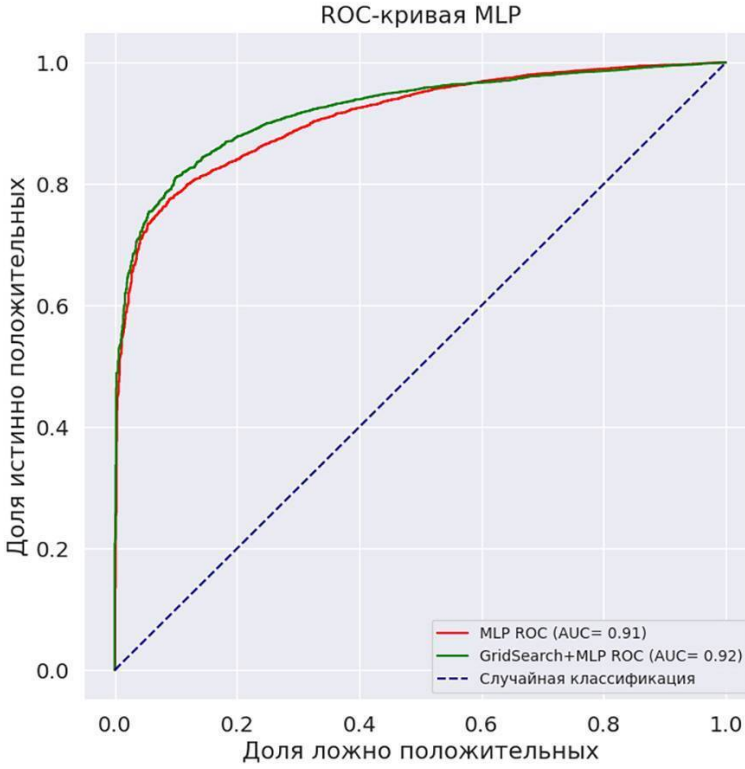
AUC: 0.92

Тем же кодом отрисуем ROC-кривые для первой и второй версий многослойного перцептрона:

```
gs_mlp_fpr, gs_mlp_tpr, gs_mlp_thresholds = roc_curve(y_test, gs_mlp_probs)
mlp_fpr, mlp_tpr, mlp_thresholds = roc_curve(y_test, mlp_probs)

plt.figure(figsize=(8,8))
plt.plot(mlp_fpr, mlp_tpr, color='red', label='MLP ROC (AUC= %0.2f)'% auc_mlp)
```

```
plt.plot(gs_mlp_fpr, gs_mlp_tpr, color='green',
label='GridSearch+MLP ROC (AUC= %0.2f)'% gs_mlp_auc)
plt.plot([0, 1], [0, 1], color='darkblue',
linestyle='–',label='random')
plt.xlabel('Доля ложно положительных')
plt.ylabel('Доля истинно положительных')
plt.title('ROC-кривая MLP')
plt.legend()
plt.show()
```

Явно возросло число итераций: кривые больше сглажены. Битва за доли процентов не прекращалась, так что прирост в 1% – это огромный прогресс.



Запустить многослойный перцептрон можно в ноутбуке. Ссылка на раздел зашита в QR-код.

Сравнение моделей

Напоследок давайте сопоставим метрики эффективности улучшенных дерева решений и многослойного перцептрона. В этом нам поможет восхитительная библиотека plotly и ее функция Figure(). Мы закладываем в нее все метрики той или иной модели. xaxis_tickangle задаст наклон названию метрики для осью x. bargroupgap – расстояние между столбиками.

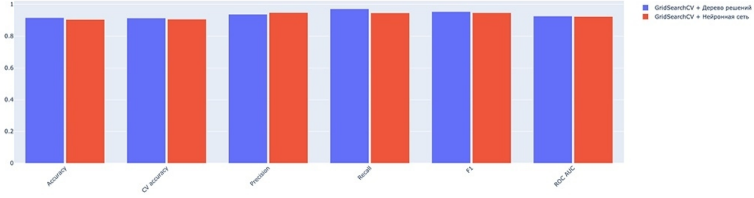
```
metrics=['Accuracy','CV
accuracy','Precision','Recall','F1','ROC AUC']
fig = go.Figure(data=[
go.Bar(name='GridSearchCV + Дерево решений', x=metrics,y=[gs_tree.score(X_test, y_test), gs_tree.best_score_, precision_score(y_test, gs_tree_pred), recall_score(y_test, gs_tree_pred), f1_score(y_test, gs_tree_pred), gs_tree_auc]),

go.Bar(name='GridSearchCV + Нейронная сеть', x=metrics,y=[gs_mlp.score(X_test,y_test), gs_mlp.best_score_, precision_score(y_test, gs_mlp_pred), recall_score(y_test, gs_mlp_pred), f1_score(y_test, gs_mlp_pred), gs_mlp_auc])])

fig.update_layout(title_text='Метрики обеих моделей', barmode='group', xaxis_tickangle=-45, bargroupgap=0.05)
```

fig.show()

Модели “идут нос к носу” в контексте пяти метрик. Выделить победителя в числовом выражении можно (это дерево решений, обозначено синим). Из этой ситуации можно сделать несколько выводов:

• для такого типа данных decision tree подходит лучше

• если не столь важны такие метрики, как accuracy, recall, F1-мера, лучше использовать MLP и опираться на AUC ROC.

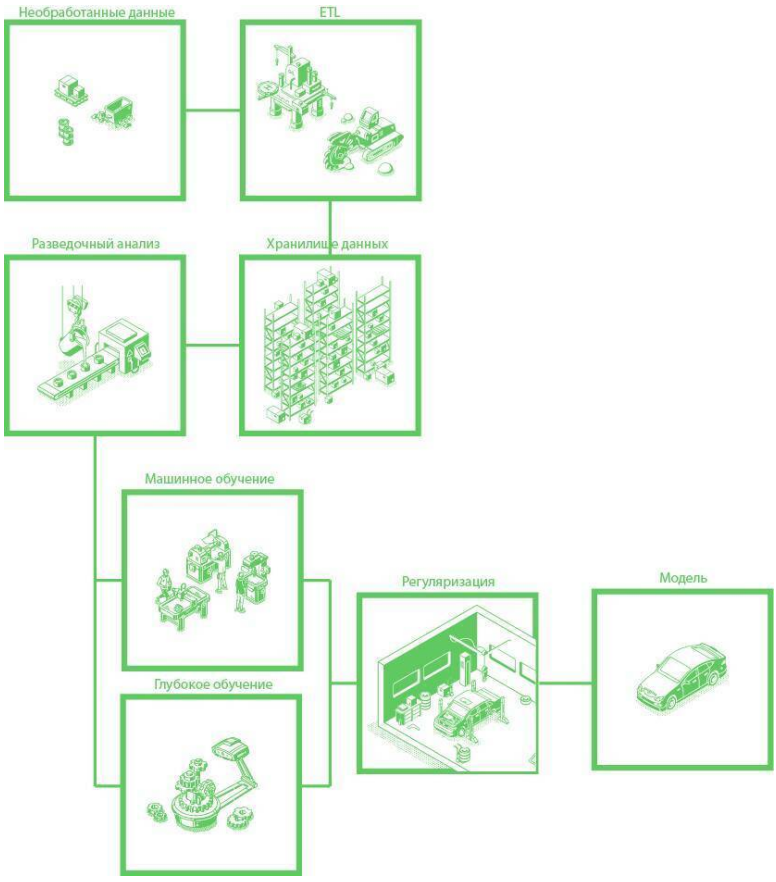


Заключение

Теперь мы знаем, что банки думают о своих клиентах в контексте продажи кредитов. В дальнейшем аналитики сопоставляют список купивших со списком тех, кто скорее всего купит. Подгруппу не купивших, но демонстрирующих признаки скорой сделки, “дожимают” рассылками, звонками и другими средствами. Мы только что решили задачу uplift-моделирования (uplift modeling), то есть помогли отделу продаж расходовать свои ресурсы допродаж эффективнее.

На этом основы машинного обучения вы восприняли, поздравляю! Пускай небольшой объём станет стимулом усвоить этот базис получше.

Вот так выглядит полноценная “конвейерная” аналогия. Надеюсь, она поможет вам быстрее запомнить, кто есть кто:



Делюсь своими любимыми ресурсами:

• towardsdatascience.com / medium.com, чтобы изучить концепции предельно доступным языком

• kaggle.com, чтобы испытать себя на соревнованиях и просто найти датасет для пробы пера.

• machinelearningmastery.com, чтобы осваивать концепции углубленно, с формулами и хорошо комментированным кодом.

• stackoverflow.com, чтобы получить ответы и предположения на самые тонкие вопросы разработки.

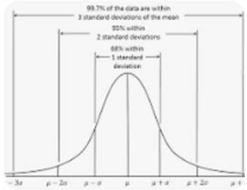
Если понятие забылось или оказалось непонятным, добивайтесь другой формы изложения с помощью поисковых систем. Google прекрасно выделяет определения из статей и собирает их в так называемые аккордеоны прямо посреди страницы с результатами.

И напоследок: сопроводив термин словом explained (например, ‘standard deviation explained’), вы на первой же странице найдете статью, которая разъясняет тему в действительно простой краткой манере. Вот так выглядят аккордеоны ‘People also ask’ («люди также спрашивают»):

People also ask

What is standard deviation for dummies?

What is standard deviation? Standard deviation tells you how spread out the data is. It is a measure of how far each observed value is from the mean. In any distribution, about 95% of values will be within 2 standard deviations of the mean. Sep 26, 2018



Cochrane
https://s4be.cochrane.org › blog › 2018/09/26 › a-beg...

A beginner's guide to standard deviation and standard error

Search for: What is standard deviation for dummies?

How do you explain standard deviation to a student?

What does a standard deviation of 3 mean?

How do you explain standard deviation example?

How do you explain standard deviation to parents?

Is standard deviation easy to understand?

Feedback