Конспект по теме "Задачи машинного обучения в бизнесе"

Что такое обучение?

Обучение состоит в поиске связей. Когда связи ищет не человек, а алгоритм, такое обучение называется машинным (сокр. ML, *machine learning*, машинное обучение).

И в бизнесе, и в быту нужно уметь прогнозировать наступление какого-либо события или *значение* какой-то величины.

Целевая переменная — та, которую предсказывают, или прогнозируют по определённым **признакам, или фичам.** Прогнозируют на основании **наблюдений** — уже известных примеров, для которых есть значения признаков и целевой переменной.

Чтобы прогнозировать успешно, вы учитесь **выстраивать взаимосвязи.** Они основаны на вашем опыте, или **эмпирических данных**. Умение выстраивать определённые взаимосвязи для прогноза называют **обучением**.

Введение в прогнозирование и машинное обучение

Обозначим набор значений целевой переменной так:

$$\vec{y}$$

Это не одно число, а набор из нескольких чисел. Такой набор называют **вектор** — отсюда и знакомая по школе стрелка. `y` — вектор целевой переменной. Для каждого объекта имеется **идентификатор уникального наблюдения**, или индекс.

Для каждого индекса `i` у вас есть набор значений признаков, или **вектор признаков.**

$$\vec{x_i} = (x_{i_1}, x_{i_2}, ..., x_{i_N})$$

, где (`x_i_1, x_i_2, ..., x_i_N`) — значения *N* параметров для *i*-го объекта.

M объектов и *N* параметров можно представить в виде таблицы, или **матрицы**. Строки — наблюдения, их ещё называют **объекты**. Столбцы — признаки. Обозначим такую **матрицу объекты-признаки** как :

$$X = \begin{pmatrix} x_{1_1} & \cdots & x_{1_N} \\ \vdots & \ddots & \vdots \\ x_{M_1} & \cdots & x_{M_N} \end{pmatrix}$$

Ваша задача — на основании этих данных вывести такую **функцию**, которая достаточно точно **отражает взаимосвязь** природных явлений и определяет зависимость между значениями признаков и целевой переменной (обозначим как ŷ — это прогноз, вычисляемый с помощью этой функции):

$$f(\vec{x}) = f(x_1, x_2, ..., x_N) = \hat{y}$$

Если применить функцию к значениям признаков отдельного объекта, получите значение — **прогноз целевой переменной для этого наблюдения.** Оно должно быть близким к тому значению популяции, которое уже наблюдали в прошлом:

$$f(\vec{x_i}) = f(x_{i_1}, x_{i_2}, ..., x_{i_N}) = \hat{y_i}$$

$$\hat{y_i} \sim y_i$$

Когда в большинстве случаев ваша формула будет давать прогноз, или **оценку** целевой переменной, близкую к её реальному историческому значению, можно сказать, что вы отыскали скрытую связь между величинами. Как же вывести эту формулу? Можно предположить, что она выглядит так:

$$\hat{y_i} = w_0 + w_1 * w_{i_1} + ... + w_N * w_{i_N}$$

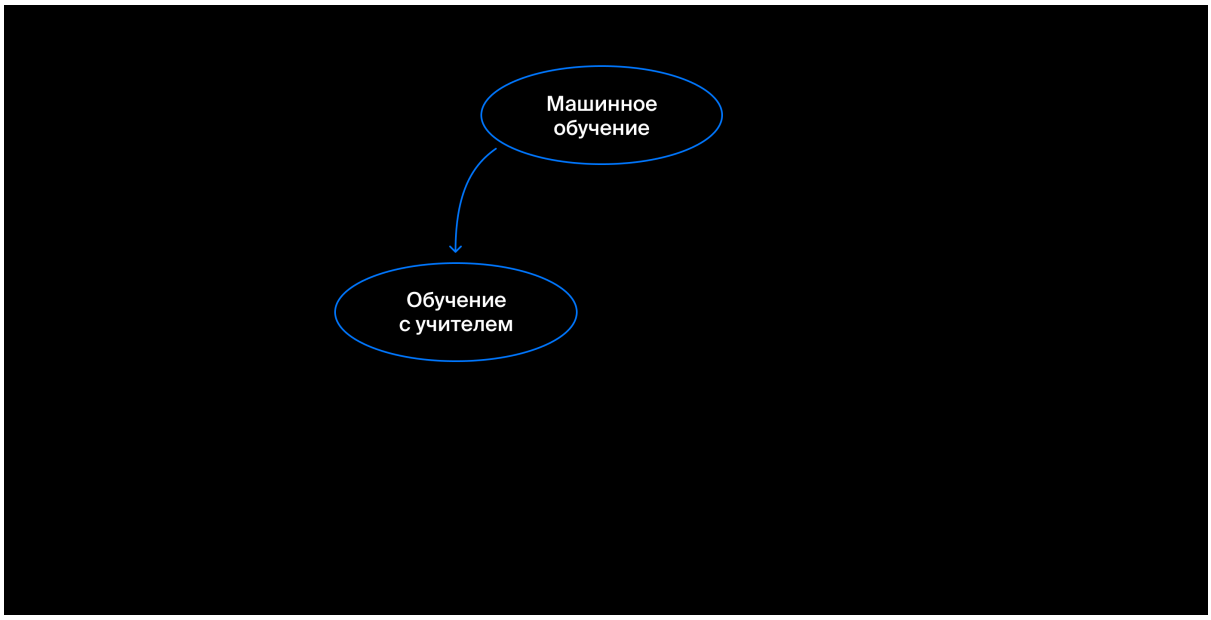
Если верить этой формуле, прогнозируемое число — это сумма «нулевого» коэффициента (когда все признаки равны 0) и значений признаков, помноженных на коэффициенты.

Допустим, с формулой угадали. Однако как подобрать такие значения `w_0, w_1, ..., w_N` (их ещё называют **параметры функции или веса)**, чтобы выведенный нами закон работал для всех наблюдений? Для этого можно делегировать поиск взаимосвязи машине (алгоритму, программе).

Машинное обучение — это **процесс поиска** машиной взаимосвязи между величинами на примере разных объектов. Обрабатывая огромные массивы данных, алгоритм **учится** на них и строит наиболее правдоподобную **модель мира**, отражающую скрытые связи процессов или объектов. Ещё надо уметь оценивать, насколько хороша модель — как точно её прогнозы соотносятся с реальностью. Подробнее вы это изучите в уроке о **метриках.** Модель обучается как человек: на основании множества наблюдений. Чем их больше — тем лучше.

Обучение с учителем

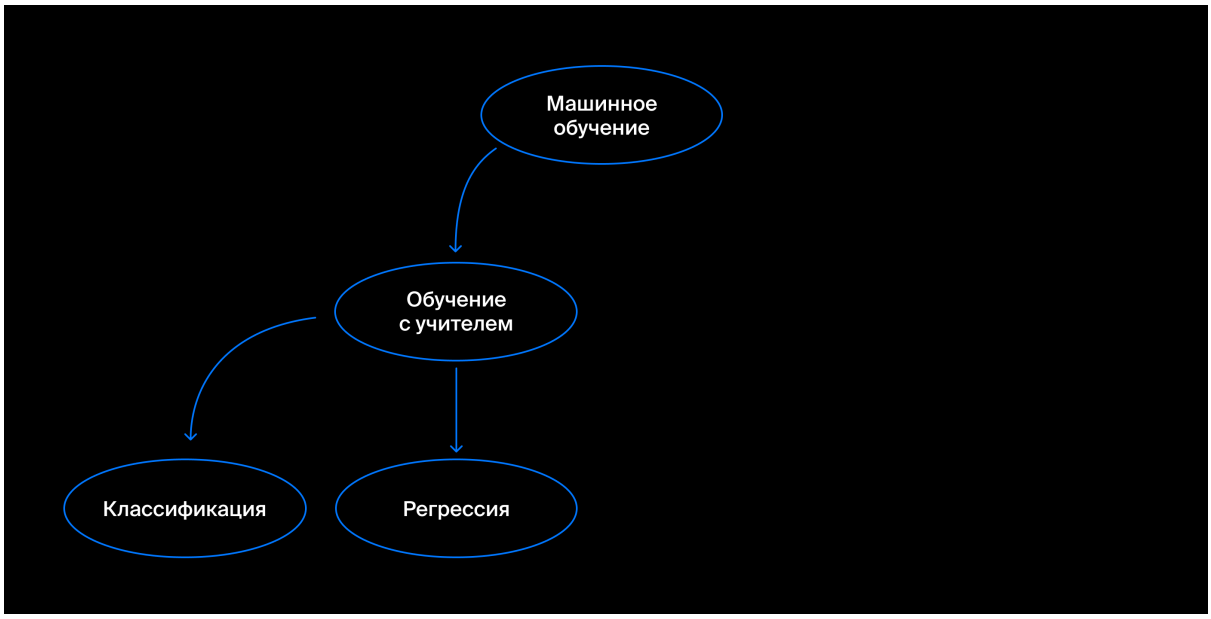
Задача из предыдущего раздела — пример **обучения с учителем** (*supervised learning*).



Моделям такого типа на вход подают большое количество наблюдений, причём для каждого известны значения признаков `x` и значение целевой переменной `y`, или **«метка» (***label*) . Поэтому говорят «на вход подают **размеченные данные**». Задача модели — восстановить зависимость между `x` и `y`, и научиться предсказывать `y` для новых объектов, у

которых известен только вектор значений признаков `X`. Здесь учитель модели — вы, поскольку определяете, что считать признаками (объектами), а что — целевой переменной, или **правильным ответом**.

Модели машинного обучения с учителем делят на **классификацию** и **регрессию**.

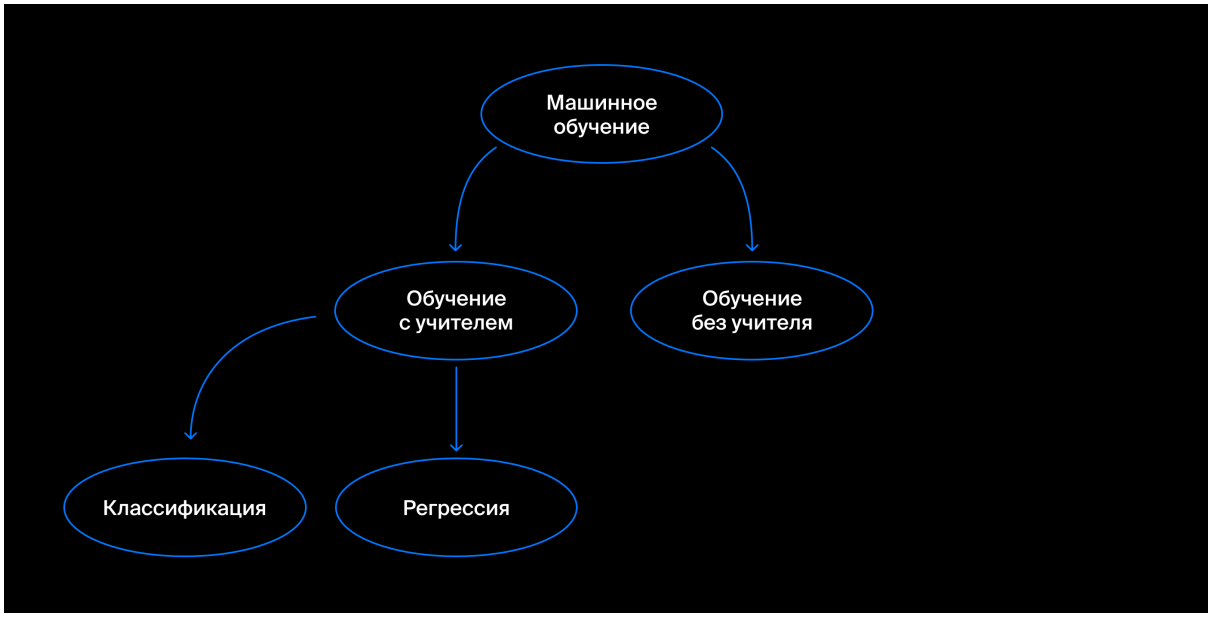


Классификацию применяют, когда в ответе модели нужно получить название **класса** каждого нового объекта. Число классов (N) может быть любым (главное, конечным и больше 1). Когда N=2, классификация будет **бинарной**. Пример задачи на бинарную классификацию — кредитный скоринг. В этой задаче нужно определить, вернёт ли заёмщик кредит или нет. Если N>2, это — **мультиклассификация**. Например, в случае если нужно сказать, вернёт ли клиент кредит в срок, вернёт с опозданием не вернёт, перед вами будет стоять как раз задача мультиклассификации. В случае **регрессии** ответ будет **непрерывной величиной**. Пример задачи регрессии — предсказание стоимости квартиры на основе её параметров.

Обучение без учителя

Вы рассмотрели задачи, для которых «истинные ответы» (*ground truth*) известны. Однако можно предоставлять модели данные без целевой переменной. Так на вход алгоритму передают большое количество

наблюдений с признаками, но без ответа (`y`), и обучают его строить взаимосвязи уже между объектами. Модель способна определить, какие объекты похожи друг на друга, а какие — нет. Такой процесс называют **обучение без учителя**. Это второй глобальный тип задач машинного обучения.



Тип задач, где нужно выявить характерные тип объектов, называют **кластеризация** — разделение объектов на группы. Главное отличие кластеризации от классификации — отсутствие заранее заданных классов, или «правильных ответов».

В кластеризации заранее заданных классов нет. Их формирует алгоритм, а не вы.

Ещё обучение без учителя применяют для **снижения размерности матрицы признаков**:

Зачем это нужно? Кажется, чем больше признаков — тем лучше. Однако это не всегда так. Размерность снижают потому, что алгоритмы не так хорошо работают, когда в условиях ограниченного объёма наблюдений признаков становится слишком много.

Последний глобальный тип алгоритмов — **обучение с подкреплением** (*reinforcment learning*). В этом случае модель обучается поэтапно и на каждом шаге немного меняет алгоритм своего действия, исходя из «подсказок» внешней среды. Обучение с подкреплением — популярная и модная область машинного обучения. Однако в этом курсе не будем в неё углубляться.

Обучение модели в Python. Библиотека sklearn

Модель — система взаимосвязей между признаками и целевой переменной или между наблюдениями, которая максимально близко отражает действительность. Модель обучают на основе уже имеющихся **наблюдений**. Строят такие модели разными методами, или **алгоритмами.** Под алгоритмом чаще всего понимают абстрактный подход к обучению

модели, однако когда дело доходит до его применения на каком-то языке программирования, говорят о **реализации алгоритма**.

В этом курсе вы рассмотрите примеры реализаций алгоритмов на языке Python — среди множества языков программирования в нём наиболее удобные библиотеки для решения задач машинного обучения. Пока хватит двух:

- `pandas` — для анализа и предобработки данных;
- `scikit-learn` — для реализации алгоритмов машинного обучения.

Обучим первую модель в Python методами библиотеки **sklearn**. В *sklearn* много инструментов работы с данными и моделей, поэтому они разложены по подразделам. В модуле **tree** находится решающее дерево. Каждой модели в *sklearn* соответствует отдельная структура данных. **DecisionTreeClassifier** — это структура данных для классификации деревом решений. Решающие деревья вы изучите позже, а пока импортируем из библиотеки структуру:

```
from sklearn.tree import DecisionTreeClassifier
```

Затем создаём объект этой структуры данных.

```
model = DecisionTreeClassifier()
```

В переменной *model* будет храниться модель. Правда, она пока не умеет предсказывать. Чтобы научилась, нужно запустить алгоритм обучения.

На вход модели передают набор значений признаков X и целевую переменную y. А у вас чаще всего есть датафрейм, где есть столбец со значениями целевой переменной и остальные столбцы. Например, у вас есть таблица `data`. В ней столбец с целевой переменной называется `'target'`. Чтобы задать матрицу объект-признак `X` и вектор целевой переменной `y`, применим метод **drop()** библиотеки *Pandas*:

```
y = data['target']
X = data.drop(['target'], axis = 1)
```

Методу передают список с названиями столбцов, которые нужно удалить. Параметр `axis = 1` указывает, что избавиться нужно именно от колонки.

Теперь уже можно построить взаимосвязь и на её основании спрогнозировать `y` по новым `X`. Чтобы запустить обучение, вызовите метод **fit()** и передайте ему как параметр данные:

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

Здесь вы передаёте на вход модели для обучения матрицу с признаками X и вектор со значениями целевой переменной.

Чтобы построить прогнозы для набора данных, хватит одной строчки кода и вызова метода **predict()**:

```
predictions = model.predict(X)
```

Тестовая, валидационная и обучающая выборки

Как оценить качество модели ?

Реальные данные, которые попадают на вход модели после её разработки, называют **тестовая выборка** (англ. *test data*). Однако перед тем как оценивать качество модели в бою, аналитики проверяют её работу на **валидационной выборке** (англ. *validation data*).

Возьмём 150 000 наблюдений. Разделим их на две неравные части — например, на 100 и 50 тысяч. Передадим первую порцию данных в модель и подождём, пока алгоритм обучится на них. Затем предложим модели предсказать ответы для второй порции и сравним, насколько прогнозы совпадают с реальными значениями целевой переменной.

Часть данных (100 тысяч), которую подают на вход модели при обучении, называют **обучающая выборка** (англ. *train data*). Те данные, на которых модель проверяют (50 тысяч), называют **валидационная выборка** (англ. *validation data*), или отложенная.

Недообучение и переобучение

Когда мы обучаем модель и «подгоняем» веса на обучающей выборке, на каждом шаге проверяем, насколько близок ответ с подобранными весами тому, что есть на самом деле. Так, ещё на этапе обучения модели, можно оценить ошибку — говорят, «**оценить ошибку на обучении**». Подобрать веса для линейной модели или другого алгоритма так, чтобы для *каждого* наблюдения был правильный ответ, почти никогда не выходит. На части наблюдений всё равно получим ошибку. Её называют **ошибка на обучающей выборке**. В процессе обучения мы стараемся эту ошибку **минимизировать**. Если сделать это не удаётся и, скажем, сколько бы ни подбирали алгоритм, он работает лишь для половины наблюдений, значит модель **недообучилась**. Возникающую в таких случаях ошибку называют **ошибка смещения** (англ. *bias*, «смещение»): ответы функции *смещены* относительно данных, потому что она не улавливает всех взаимосвязей. Вот отчего так бывает:

- Слишком мало примеров или признаков;
- Слишком простая функция;
- Неверный подход к подбору разных вариантов искомой зависимости;

Такая модель будет давать одинаково плохие результаты и на обучающей, и на тестовой выборке.

В идеальном случае модель (функция, алгоритм) должна не только редко ошибаться на обучении, но и хорошо работать на новых данных, которые она не «видела» при подгонке весов или поиске оптимальной зависимости. Говорят, что модели нужно иметь хорошую **обобщающую способность**. Тогда от применения машинного обучения будет польза.

Представим модель, которая идеально точно предсказывает значения на тестовой выборке. Можно сказать, что ваша модель «идеальна», но лишь до тех пор, пока ей на вход не передадут значения признаков для новых объектов. Когда на валидационных данных модель даёт результаты значительно хуже, чем на обучающих, говорят, что она **переобучилась** (англ. *overfitting*). Такую ошибку называют **ошибка разброса** (англ. *variance*). Это значит, что модель слишком сильно подстраивается под данные с учётом их шума: при переобучении она учитывает не только действительные связи в данных, но и избыточную информацию.

Разделяй и валидируй

В быту аналитики часто называют валидационную выборку тестовой. В scikit-learn функция, разделяющая выборку, называется `train_test_split()`. Это может вводить в заблуждение, но не поддавайтесь панике: постарайтесь понять, что имеют в виду в каждом случае.

Как бы вы ни называли разделённые выборки, фундаментальными остаются два вопроса:

- **В каких пропорциях делить?** Часто делят в соотношении 80 к 20 или 70 к 30. Общее правило: обучающая выборка должна быть больше отложенной, но так, чтобы проверка метрик на тесте осталась обоснованной.
- **Каким способом делить?** Вы ещё узнаете, что есть два способа делить выборку на обучающую и валидационную: **по времени** и **случайно**. Первый годится, когда более ранние наблюдения влияют на более поздние. Второй — когда временная структура не так важна для обучения модели.

Быстрый способ — вызвать функцию `train_test_split()` из модуля `model_selection` библиотеки `sklearn`. Синтаксис такой:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Здесь вы передаёте на вход функции матрицу признаков `X`, вектор с целевой переменной `y`, а также параметр `test_size` — он сообщает, какую долю выборки оставить на валидацию. Функция возвращает две матрицы признаков и два вектора с целевой переменной, полученные разделением исходных в пропорции, указанной в `test_size`.

По умолчанию `train_test_split()` делит выборку в этой пропорции случайным образом: если запустить код несколько раз, полученные матрицы признаков и векторы с целевой переменной будут разными. Чтобы ваши результаты во время учёбы были сопоставимы с результатами других студентов, при разделении выборки на обучающую и валидационную указывайте параметр `random_state` с нулевым значением:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
tate=0)
```

Параметр `random_state` встречается и в других функциях, где тоже отвечает за «случайность». Например, при задании алгоритма для модели:

```
model = RandomForestRegressor(random_state=0)
```

Оценка качества модели

Когда вы разделили выборку правильным способом и в нужной пропорции, обучите модель на обучающей (*train*) выборке и оцените её качество на валидационной (*validation*).

Рассмотрим синтаксис метрик качества на примере **коэффициента детерминации**, или **R-квадрата**. Так называют часть дисперсии (изменчивости) целевой переменной, которую объясняет модель. Эта метрика принимает значения от 0 до 1 и показывает, насколько сделанный моделью прогноз отражает целевую переменную. Если модель объясняет идеально (делает максимально точный прогноз), значение коэффициента детерминации будет равно 1. А если модель прогнозирует из рук вон плохо, R-квадрат — 0. Формула выборочной оценки коэффициента детерминации выглядит так:

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

Выглядит как всегда пугающе, но смысл довольно простой. В числителе дроби суммарная квадратичная ошибка, поэтому если вы почти всегда даёте близкий прогноз, дробь будет равна 0, а значение метрики — 1. В знаменателе суммарная разница между значением и средним. Это нужно, чтобы нормировать вашу ошибку на фактический разброс целевой переменной. Если вы сильно ошибаетесь, да и сама величина очень изменчивая, это немного скомпенсирует разницу между прогнозом и фактом.

Синтаксис R-квадрата (`r2_score`) простой — в качестве параметров вы передаёте реальный и спрогнозированный вектора целевой переменной:

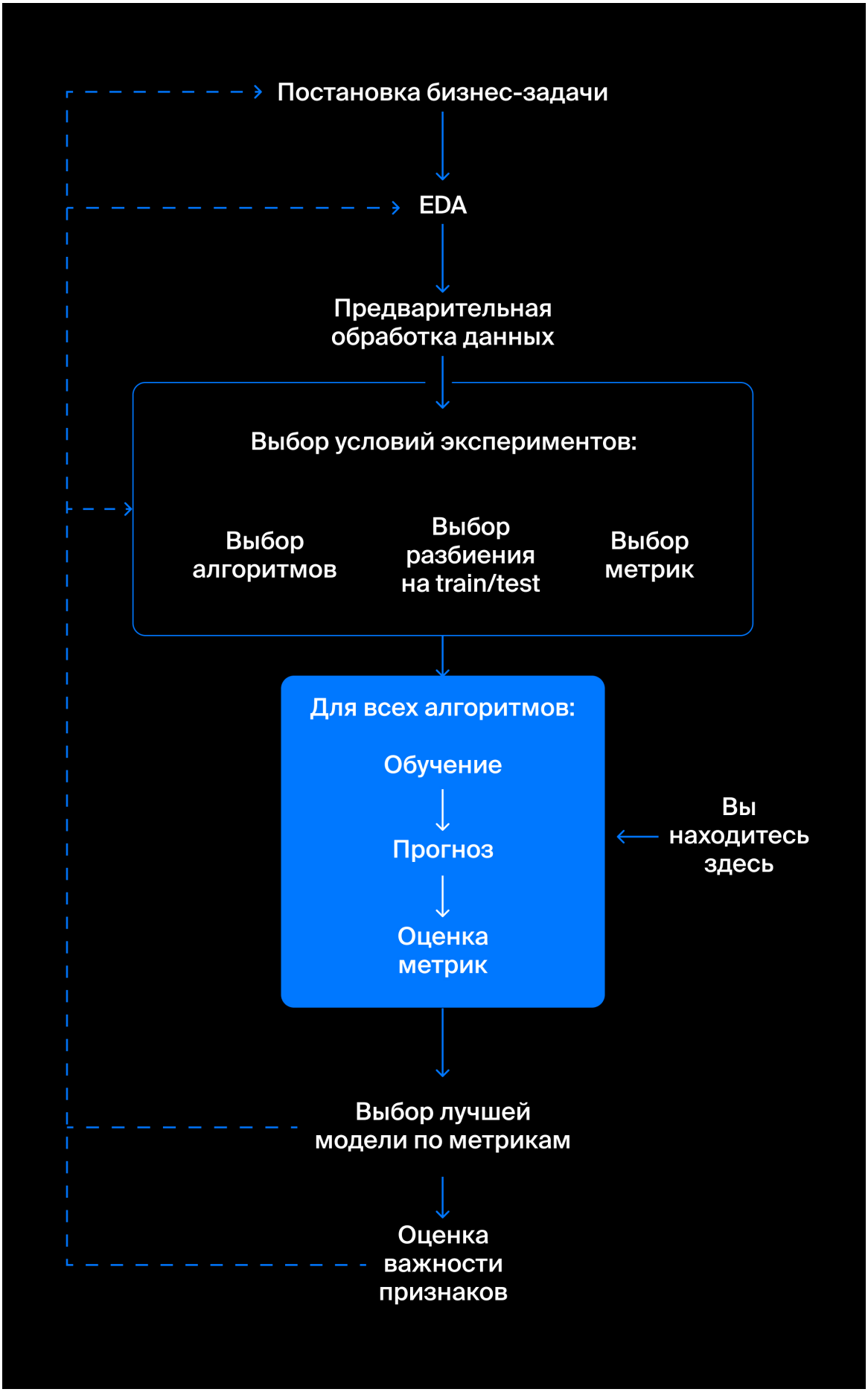
```
metrics.r2_score(y_train, y_pred)
```

Эта функция возвращает одно значение, равное коэффициенту детерминации.

Пайплайн машинного обучения

За модным словосочетанием «машинное обучение» скрывается простая концепция. Есть выборка из наблюдений, для которых вам известны значения признаков (вектор признаков `X`) и иногда значение целевой переменной — `y`. Затем выборку нужно разделить: на одной части данных — обучить модель, на второй — проверить, а прогнозировать — на третьей. На практике аналитик, желающий обучить модель, сталкивается с дополнительными задачами.

Вот базовые этапы работы над задачей машинного обучения:


Постановка бизнес-задачи
EDA
Предварительная обработка данных
Выбор условий экспериментов:
Выбор алгоритмов
Выбор разбиения на train/test
Выбор метрик
Для всех алгоритмов:
Обучение
Прогноз
Оценка метрик
Вы находитесь здесь
Выбор лучшей модели по метрикам
Оценка важности признаков

В зависимости от задачи (обучение с учителем или без, задача регрессии или классификации) и особенностей алгоритмов, последовательности шагов могут быть разными. Однако есть общие базовые принципы, достойные вашего внимания.

Постановка задачи

Определяет дальнейшую судьбу вашей модели. На этом этапе вы переводите определённую бизнес-задачу в плоскость математики, аналитических инструментов и машинного обучения. Убедитесь, что вы хорошо понимаете стоящую перед бизнесом проблему — от этого зависит выбор модели, алгоритма и метрик.

EDA

Прежде чем отправить данные на вход модели и получить прогнозы, проводят «разведку данных» — **EDA** (англ. *exploratory data analysis*), или **исследовательский анализ данных**. На этом этапе вы изучаете распределения отдельных признаков и целевой переменной, строите корреляции между величинами, исследуете специфику датасета. Бывает полезно построить гистограммы признаков. В библиотеке *seaborn* такие графики отрисовывают функцией **distplot()**:

```
import seaborn as sns

sns.distplot(df['feature_1'])
```

Разведка данных позволит сформулировать первые гипотезы относительно качества данных и аномалий в них. Уже на этом шаге можно предположить, какие признаки станут ключевыми для модели, а какими можно и даже нужно пренебречь. EDA способен вдохновить вас на генерацию новых признаков на основе имеющихся. Такой анализ важен с точки зрения как бизнеса, так и качества будущей модели. Вы погрузитесь в природу данных, увидите за цифрами смысл и сделаете ценные предположения.

Подготовка данных

Когда в вашем распоряжении пара часов, а построить базовую модель (англ. *baseline*, «базовый уровень») нужно срочно, без подготовки данных

можно обойтись. Однако на практике данные часто **предобрабатывают**, а именно:

- обрабатывают пропущенные значения;
- преобразуют отдельные признаки (например, переводят категориальные признаки в количественные);
- нормализуют или стандартизируют данные (эту операцию подробно разберём в следующей теме);
- создают новые признаки (фичи) на основе уже существующих. Этот этап ещё называют **feature engineering** (англ. «проектирование признаков»), и иногда он может вывести вашу модель на совершенно новый уровень качества.

Выбор валидационной стратегии

В прошлом уроке вы разделили выборку на обучающую и валидационную и узнали, почему важно оценивать метрики именно на отложенной части наблюдений. В зависимости от типа данных и особенностей задачи, на этом этапе следует выбрать способ формирования валидационного множества. Вы изучите это подробнее в третьей теме курса.

Также на этом шаге важно убедиться, что распределение величин на обучающей выборке близко к тому, с которым модель будет работать в реальности. Иначе всё это не имеет смысла.

Выбор алгоритма

В зависимости от типа задач (с учителем или без) и поставленной проблемы, у вас есть целый арсенал различных алгоритмов со своими достоинствами и недостатками. Одни алгоритмы точнее, но их сложно интерпретировать; другие — быстрые, но притом слабее. Вот основные критерии выбора алгоритма:

- Точность;
- Скорость;
- Интерпретируемость;
- Индивидуальные особенности алгоритмов: на разных типах признаков они работают по-разному.

Даже у самого простого алгоритма есть множество настраиваемых параметров. Иногда они могут повлиять на качество и скорость обучения вашей модели. Чаще всего выбор параметров происходит итеративно. Вы обучаете модель с одними параметрами, оцениваете метрики, видите, что они так себе — меняете параметры, снова обучаете и проверяете качество. Это можно делать бесконечно, но мы научим вас останавливаться вовремя.

Выбор метрик

Прежде чем обучать алгоритм, определитесь, как будете оценивать его качество.

Для каждого типа задач (классификация, регрессия, кластеризация) есть стандартный набор метрик. Однако важно не просто «прогнать» результаты через этот набор, а понять, какая метрика лучше всего отражает суть бизнес-процесса.

Во второй теме вы узнаете, какие бывают метрики и чем они отличаются. Когда-нибудь вы даже придумаете свою метрику, чтобы убедить коллег, что модель действительно помогает бизнесу и вообще приносит пользу.

На этом же шаге полезно узнать, какие способы решения задачи уже применяют в компании. Это позволит честно оценить, действительно ли машинное обучение даёт вам преимущество в сравнении с более консервативными инструментами (*baseline*).

Обучение и прогнозирование

Вы прошли все шаги, дальше вступает алгоритм с традиционным `fit-predict`. На стадии *fit* вы передаёте ему подготовленную порцию данных (*train*-выборку), он на ней обучается и строит взаимосвязи между признаками.

Переходим к *predict*. У вас осталась отложенная порция данных, для которых вы знаете признаки и ответы. На этом этапе вы берёте только признаки, передаёте их на вход обученной модели и сохраняете предсказанные значения.

Оценка качества результатов и выбор лучшей модели

На этом шаге вы определяете, насколько спрогнозированные вами значения для объектов из валидационной выборки отличаются от реальных. Часто оценивают не один, а несколько алгоритмов и на основании выбранных метрик выбирают лучший.

Оценка важности признаков

Вы выбрали самый успешный алгоритм, который показывает лучшие в сравнении с *baseline* и другими алгоритмами результат. Чаще всего этого недостаточно, чтобы начать использовать вашу модель в бою. Нужно ещё раз убедиться, что модель отразила правильные паттерны и взаимосвязи между данными. Как это сделать? Например, применив **анализ важности признаков**. Этот набор подходов позволяет оценить не только *что* предсказала модель, но и *почему*. Оценка важности признаков позволяет коллегам из других подразделений поверить в модель и взять её на вооружение.

Что дальше?

Вы нашли данные, преобразовали их, разделили на обучающую и валидационную выборки, выбрали несколько подходящих алгоритмов. Дальше `fit-predict`. Выбрали лучшую модель, проинтерпретировали важность признаков и даже сами поверили в то, что она работает. Вы прекрасны! Внедряем модель в текущие рабочие процессы и реализуем в продуктивной среде?

На самом деле всё не так просто: вы проходите пайплайн один раз, а потом чаще всего возвращаетесь на предыдущие этапы, что-то меняете и смотрите, как преобразился результат. Это совершенно нормально. Как нормально и то, что иногда, как бы хороши ни были вы, данные и модель, в применении машинного обучения смысла никакого.

Почему машинное обучение — не панацея?

Вы познакомились с машинным обучением. Узнаем не менее важную вещь — в каких случаях применять его *не надо*. Прежде чем лихо отправить данные в модель машинного обучения, ответьте на важные вопросы:

- достаточно ли у вас большая выборка?

- насколько качественные данные?
- способна ли ваша модель дать правдоподобный прогноз?

Выборка

Однозначного ответа на вопрос, что считать достаточно большой выборкой, нет. Это зависит от каждого конкретного случая. Часто, определяя размер выборки, основываются на **эмпирических правилах**, то есть выведенных из опыта. Они учитывают количество признаков, разнообразие значений целевой переменной и специфику самих алгоритмов.

Первое правило гласит, что минимально необходимое количество наблюдений в выборке линейно связано с **числом признаков**. То есть минимальная выборка может быть рассчитана по формуле `s = k * n`, где `n` — число признаков для каждого наблюдения, а `k` — коэффициент, который из опыта часто считают равным 10. Если на одного пользователя приходится 20 признаков, то для обнаружения зависимостей нужно как минимум 200 пользователей. А вот если признаков 150, то и тысячи пользователей не хватит.

Другое эмпирическое правило подходит для задач кластеризации и предлагает отталкиваться от **количества целевых кластеров.** Чем больше кластеров, тем сложнее научиться различать их на основании доступных признаков. Если рассчитывать минимально необходимое число наблюдений по предыдущему эмпирическому правилу и увеличивать при этом в `n` раз число целевых классов, придётся увеличить полученное значение во столько же раз.

Можно ли «поделить» 400 пользователей на 10 кластеров, когда в признаках у вас 234 показателя? Скорее всего, вы столкнётесь с **проклятием размерности**. Числа наблюдений будет недостаточно, чтобы сделать группировку в пространстве всего множества признаков.

Последнее правило предлагает отталкиваться от **семейства алгоритмов**. Например, нейронные сети плохо работают на малых данных. Им подавай выборки в сотни тысяч наблюдений.

Общий вывод такой: если число клиентов вашего бизнеса или наблюдений не выходит на порядки тысяч, машинное обучение не слишком нужно. Ваша сила — в индивидуальном подходе и эксклюзивности!

Качество данных

Допустим, у вас миллионы наблюдений. Такой миллионер — мечта любой модели! Но всегда ли количество решает? Что если на самом деле ваши данные не несут ожидаемого объёма информации, чтобы выстроить хотя бы какие-то взаимосвязи?

В машинном обучении работает правило **GIGO** (англ. *Garbage In, Garbage Out*, «Мусор на входе — мусор на выходе»)**.** Если на вход модели передать некачественные данные, даже при правильном выборе алгоритма на выходе получите неверные результаты. Вот потенциальные проблемы с качеством данных:

- шум;
- пропуски;
- ошибки и выбросы;
- изменение в распределении данных со временем.

На этапе EDA узнайте, есть ли такие проблемы в вашем датасете. Часть из них можно решить методами предобработки.

Однако бывает, что вы не можете повлиять на качество данных. Например, если для какого-то признака доля пропущенных значений больше половины и наблюдения не связаны во времени — не выйдет «протянуть» значение с предыдущих периодов — скорее всего, придётся избавиться от такой переменной.

Ещё один пример из практики — **изменчивость данных.** Важное правило применимости моделей машинного обучения: распределения признаков на обучающей, валидационной и тестовой выборках должны быть похожими. Иначе модель будет просто бесполезной.

Низкое качество модели

И такое бывает. Казалось бы и данных много, и качество у них отличное, но метрики сообщают, что на основании имеющихся данных вы не можете прогнозировать выбранную целевую переменную. Причина в том, что все ваши признаки никак не связаны с вашей целевой переменной, и, соответственно, по ним прогнозировать интересующую вас величину нельзя.

Так, данные о продажах в салонах российского оператора сотовой связи никак не помогут вам для прогнозирования популяции пингвинов в ЮАР, какими бы точными они ни были. Скорее всего, вы получите относительные ошибки на уровне сотен процентов, а R-квадрат будет близким к 0.

Другой пример — очень шумные, или волатильные данные. Среди таких почти невозможно уловить «полезный сигнал». В таких случаях машинное обучение не годится, и применяют другие методы математического моделирования.