Конспект по теме "Процесс решения задач машинного обучения"

Постановка задачи

Формулирование бизнес-задачи

Основные вопросы, на которые важно ответить:

- **Что мы прогнозируем и какую бизнес-задачу этим решаем?**
- **Какие данные для этого есть?**
- **Кто и как будет использовать вашу модель (как она встраивается в бизнес-процесс)?**
- **Какие результаты вы хотите получить?**
  - как в компании до вас решали эту проблему и насколько успешно?
  - какой бизнес-эффект может дать ваша модель?
  - достаточно ли у вас ресурсов (время, люди, финансы, вычислительные ресурсы)?
  - по каким метрикам вы будете оценивать качество работы вашей модели?
  - есть ли успешные бенчмарки (примеры с рынка) использования машинного обучения в подобных задачах?

Формализация задачи и её перевод на язык машинного обучения

Если вы смогли чётко ответить на все эти вопросы и убедиться в том, что вам действительно нужно использовать инструменты машинного обучения, переводите бизнес-постановку задачи в термины ML. Что это значит:

- определение типа задачи (обучение с учителем/без, классификации/регрессии/кластеризации)

- признаки:
  - какие данные выбрать в качестве признаков?
  - каков размер выборки?
  - за какой период?
  - каково качество этих данных?
  - есть ли в данных временная структура?
- какие метрики вы будете использовать при оптимизации и оценке качества модели
- есть ли у модели ограничения?
  - скорость
  - интерпретируемость результатов
  - точность
  - время на разработку
- какие алгоритмы вы планируете использовать? На этот вопрос вы сможете ответить с учётом предыдущих пунктов

EDA. Анализ качества признаков

Когда вы определились с задачей и тем, какие данные вы будете использовать, а самое главное, действительно заполучили их (например, в виде csv-файла), следующий этап — посмотреть на эти данные и, возможно, сразу выдвинуть ряд гипотез об особенностях процесса, который вы моделируете, или о том, какие признаки для него важнее всего. Это называется **exploratory data analysis** (первоначальный анализ данных или, как ещё говорят, «разведывательный» анализ). Задача такой «разведки» данных:

1. оценить их качество и объём необходимой работы по предобработке
2. посмотреть на распределения и взаимные корреляции и выявить какие-то аномалии, если они есть
3. сформулировать первые гипотезы относительно признаков или целевой переменной

Основные вопросы, на которые полезно ответить на этом этапе оценки качества данных:

- какой размер датасета?
- какие в него входят признаки? Если они не обезличены (а такое бывает чаще всего только на специальных конкурсах), то лучше предварительно изучить, что значит каждый признак, чтобы глубже погрузиться в суть бизнес-процесса.
- каков тип ваших признаков? Они чаще всего бывают **числовые и категориальные.**
- имеет ли целевая переменная временную структуру (стоит ли задача прогнозирования временного ряда)? В зависимости от этого на следующих этапах вы можете применять разное разделение на обучающую выборку и валидационную выборку, а также использовать специфические производные признаки на основе исходных.
- какова доля пропущенных значений? Иногда вы можете сразу отметить для себя, что какие-то признаки очень полезны, но, к сожалению, они присутствуют только для 1% наблюдений и вряд ли можно делать на них ставку в модели. Здесь вы можете решить, как работать с пустыми значениями — или выбрасывать их из датасета, или заполнять из «прошлого» или «из будущего».

EDA. Формулировка гипотез

Прикинув, что у вас за признаки, можете ещё внимательнее посмотреть на них. Полезно:

- посмотреть на распределение числовых признаков. Постройте для них гистограммы. Так вы увидите, есть ли какие-то аномальные выбросы в значениях, и как вообще ведёт себя признак. Желательно, если бы каждый из них был более-менее нормальным
- если у вас есть привязка к дате, нарисовать графики распределения величин во времени. Особенно при работе с производственными процессами. Так вы сможете понять, менялось ли поведение признака со временем. В самой первой теме мы говорили, что имеет смысл использовать модель, только когда вы уверены, что признаки будут

вести себя примерно как раньше — иначе вся ваша модель станет бесполезной

- по распределению признаков и целевой переменной оценить наличие выбросов — они сразу бросаются в глаза на гистограммах и других графиках
- рассчитать матрицу корреляций и на её основе построить heatmap

  Расчёт матрицы корреляций методом датафреймов *corr()* занимает одну строчку кода:

  ```
  cm = df.corr()
  ```

  Теперь в переменной cm хранится матрица корреляций. Чтобы представить её визуально, воспользуйтесь знакомым вам методом *heatmap()* библиотеки *seaborn*:

  ```
  sns.heatmap(cm, annot = True, square=True)
  ```

  На этом этапе уже можете посмотреть:

  - какие признаки наиболее сильно коррелируют с целевой переменной
  - какие признаки сильно коррелируют между собой. Вы же помните, что для линейных моделей взаимная корреляция нежелательна, поэтому при выборе такой модели уделите этому особое внимание
- построение попарных графиков (бывает полезным в дополнении к предыдущему пункту, потому что корреляция отражает только линейную зависимость, а на попарных графиках можно увидеть другие взаимосвязи):
  - признак-признак
  - признак-целевая переменная

  Отрисовка попарных графиков производится методом *scatterplot()* библиотеки *seaborn,* например, так:

  ```
  sns.scatterplot(df['Признак 1'], df['Признак 2'])
  ```

Если вы достаточно долго помедитировали над графиками, наверняка сможете выдвинуть предварительные гипотезы:

- какие признаки могут быть наиболее ценными для модели исходя из корреляций
- какие полезные признаки можно попробовать сгенерировать самостоятельно на основе уже имеющихся
- насколько полезной вообще может быть модель с таким качеством исходных данных. Если многие признаки слишком шумны и изменчивы, уже в самом начале важно это признать и не ждать, что ML совершит чудо.

Предварительная обработка данных

Когда вы посмотрели на данные, оценили масштаб бедствия (количество пропусков, выбросов и категориальных признаков), вы уже можете сказать, какую именно предобработку сделать, чтобы применить тот алгоритм, который выбран для обучения модели.

Обычно предврительная обработка данных включает следующие этапы:

1. Работа с пропущенными значениями
2. Работа с выбросами
3. Преобразование категориальных переменных
4. Стандартизация данных в случае работы с линейными моделями или моделями, которые чувствительны к расстоянию (например, кластеризация)

Иногда к этому же этапу относят отбор признаков, но это можно делать и на более поздних стадиях разработки модели.

Разберём подробнее каждый пункт.

Работа с пропущенными значениями

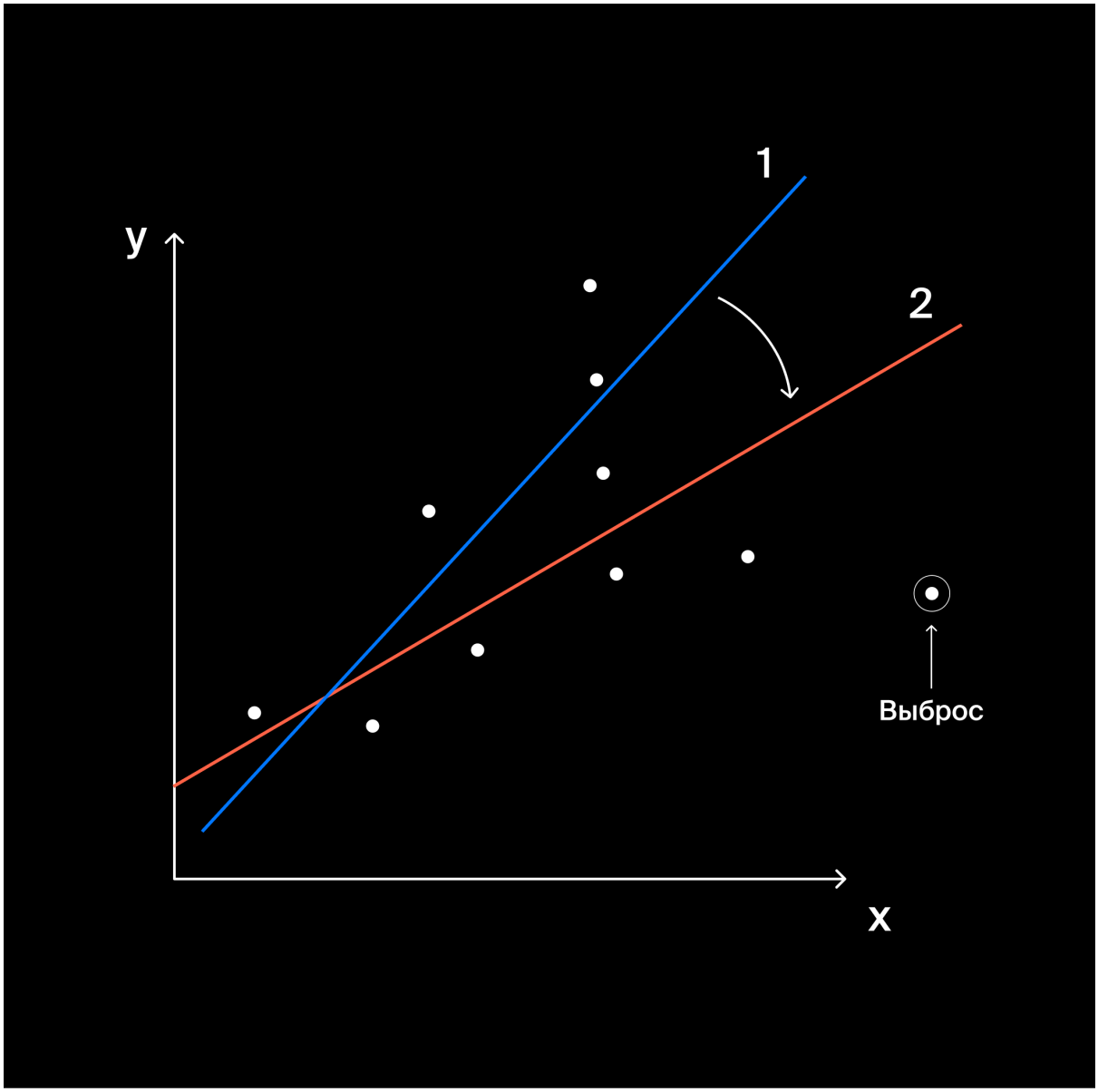
Есть несколько способов работы с пропусками. В основном они сводятся к следующим:

- удаление наблюдений с пропущенными значениями. Работает, когда данных очень много, а пустых значений, напротив, не очень. Тогда можете пожертвовать малой частью наблюдений, в которых пропущены значения отдельных признаков. Но, вообще, это, конечно, достаточно радикальный подход — не пытайтесь повторить это в домашних условиях
- заполнение соседними значениями «из прошлого». В курсе по обработке данных вы работали с методом `fillna()`. Когда у вас есть привязка ко времени и один из параметров присутствует в данных с меньшей частотой, чем другие, тогда применяется `fillna()` с параметром `'method'='ffill'`. Здесь мы справедливо предполагаем, что отсутствующие значения стоит заполнять значениями «из прошлого»
- заполнение средними значениями. Этот метод, напротив, характерен для признаков, у которых нет временной структуры и не за что «зацепиться». То есть если для какого-то наблюдения отсутствует значение одного из признаков, можете предположить, что оно среднее (чтобы это не сильно повлияло на прогноз для него) или медианное
- заполнение нулями. Тоже «радикальный» метод. Особенно осторожно применяйте его, если собираетесь работать с линейными алгоритмами — для них нулевое значение может поломать всю линейную зависимость. Деревья сработают с таким заполнением лучше, так как при делении выборки смогут учесть факт незаполненности данного признака, если оно связано с целевой переменной
- иногда (особенно в случае категориальных признаков) мы можем заменить пропуски индикатором «незаполненного» значения. Иногда тот факт, что человек не вписал в анкету «свой» город, может что-то значить — и правильнее будет не заполнять его наиболее часто встречающимся городом.

Общий подход к пропущенным значениям — попробуйте аккуратно поработать с ними, но при обучении модели следите, не приводят ли заполненные значения к аномалиям. Если так, возможно, от признаков с большим количеством пустых значений придётся избавляться.

Работа с выбросами

Во многом похожа на работу с пропусками. Бóльшая часть алгоритмов устойчива к выбросам, но некоторые нет. Например, линейные модели — выбросы могут серьёзно «перетянуть» зависимость на себя. Рассмотрим два облака точек, отличающиеся только двумя наблюдениями. Но смотрите, как сдвигается предложенная модель — всего одна точка меняет предложенную моделью зависимость с варианта 1 на вариант 2:



Стандартная последовательность работы с выбросами:

- выделение порога, при котором наблюдение считается выбросом, — например, где признак (или целевая переменная) находится ниже 5%-квантили или выше 95%.

- удаление (подходит для больших выборок и небольшой доли выбросов) таких наблюдений из выборки или замена значений средним либо максимальным/минимальным значением для этого признака.

Преобразование категориальных переменных

Чаще всего преобразование категориальных переменных может быть двух типов:

1. перевод в числовые значения без существенных преобразований. Концептуально существуют два подхода:
   - замена категорий числовыми лейблами (поэтому такое преобразование называют *label encoding*).

     Например, преобразование **строковых** значений признака «Город» из «Сургут», «Москва», «Санкт-Петербург» в **числовые** 0, 1 и 2, соответственно. Для этого предназначен класс *LabelEncoder()* из модуля *sklearn.preprocessing*:

     ```
     from sklearn.preprocessing import LabelEncoder

     print(df['column'].head())

     encoder =  LabelEncoder() # создаём переменную класса LabelEncoder - коди
     ровщик
     df['column'] = encoder.fit_transform(df['column']) # используем кодировщи
     к, чтобы "перевести" строковые названия в числа

     print(df['column'].head())
     ```

     Теперь признак «Город» — числовой, а не категориальный. Но у него есть существенный недостаток — он устанавливает между категориями отношения типа «больше»/«меньше». Для моделей, основанных на линейной взаимосвязи (например, для линейной регрессии), это не очень хорошо. Поэтому чаще используются другие подходы.

   - Представление одного категориального поля в множество бинарных полей (такое преобразование называют *one-hot encoding*).

     В этом случае вместо поля «Город» появятся поля «Москва», «Санкт-Петербург», «Сургут» и другие, а эти новые признаки

будут принимать только значения 0 или 1. Для этого существует функция *pandas.get_dummies()*. Ей на вход передаётся весь датафрейм, а она сама выделяет категориальные переменные, преобразует их в новые признаки (с удобными новыми названиями), которые называют **dummy-переменными**, и возвращает обновлённый датафрейм:

```
print(df.head())
df = pd.get_dummies(df)
print(df.head())
```

Для такого преобразования также можно использовать класс *OneHotEncoder* модуля *sklearn.preprocessing*, но он чуть менее удобен.

Бинарная замена категориальных признаков хороша почти всегда. Но если их слишком много или у какого-то одного из них много уникальных значений (или сразу то и другое), преобразование сильно раздувает матрицу (датафрейм). Почти у всех моделей машинного обучения с этим большие проблемы (не говоря уже о том, что это увеличивает время всех расчётов) — тут вспоминают про label encoding или просто избавляются от отдельных признаков.

2. Более сложные вычисления новых признаков на базе существующих категориальных. Например, создание признака «Пол» по значению признака «ФИО». Или если признак представляет собой текстовый комментарий, можно сгенерировать много разных числовых характеристик этого текста — длина сообщения, факт употребления стоп-слов, число повторений слов, негативный или позитивный окрас и другие.

4. Стандартизация данных

Многие алгоритмы машинного обучения лучше работают на стандартизированных данных, где значения признаков соответствуют стандартному нормальному распределению. Стандартизация совершенно **необходима** в двух областях ML:

- линейная регрессия;
- кластеризация и методы, строящиеся на взаимных расстояниях между объектами.

Процедура приведения конкретного признака к стандартизированному виду выглядит так:

- вычисляется среднее значение признака в выборке и вычитается из каждого наблюдения (чтобы значения «центрировать» около 0);
- каждое полученное значение делится на стандартное отклонение (чтобы масштабировать исходный «разброс» значений к 1).

Поэтому в *pipeline* разработки модели стандартизация выглядит следующим образом:

- разделить выборку на *train* и *test*:

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

  Здесь вы видите классическое деление на обучающую и валидационную выборку в пропорции 80:20.

- обучить и применить «стандартизатор» на обучающей выборке:

  ```
  scaler = StandardScaler()
  X_train_st = scaler.fit_transform(X_train)
  ```

  Здесь «стандартизатор» запомнил среднее и стандартное отклонение вашей обучающей выборки, а также применил стандартизацию с учётом этих «знаний» к ней.

- применить «стандартизатор» для тестовой выборки

  ```
  X_test_st = scaler.transform(X_test)
  ```

Здесь вы стандартизируете тестовые данные и сохраняете в переменной X_test_st матрицу (не датафрейм) с трансформированными значениями признаков. Но имейте в виду, что трансформируете вы значения в тестовой выборке с учётом среднего значения и стандартного отклонения, которые были вычислены на другой выборке — обучающей!

Кажется, это противоречит логике, и после такой трансформации распределение изменённых тестовых данных все равно может отличаться от стандартного нормального распределения. И порой это

действительно так. Тому есть 2 причины: во-первых, при тестировании модели в реальной жизни вы тоже не знаете заранее будущее распределение значений ваших признаков; а во-вторых, если распределения в тестовой выборке (а значит, среднее значение и стандартное отклонение) очень сильно отличаются от того, что было на обучающей, ваша модель и так будет работать не очень хорошо.

Random и time split

После предварительной обработки данных вашу модель уже можно в принципе обучить каким-нибудь алгоритмом. Но вы хотите не просто построить модель, а выбрать из нескольких лучшую. Это делается сравнением метрик для различных моделей на валидации (валидационной выборке). Здесь важно ответить на три вопроса:

1. Как вы будете делить выборку, чтобы оценивать качество на валидации?
2. Из каких моделей будете выбирать?
3. На основании каких метрик выберете модель?

В этом уроке вы узнаете, как ответить на первый вопрос.

Как делить выборку?

Принципиально существует два подхода:

- случайно
- с учётом времени

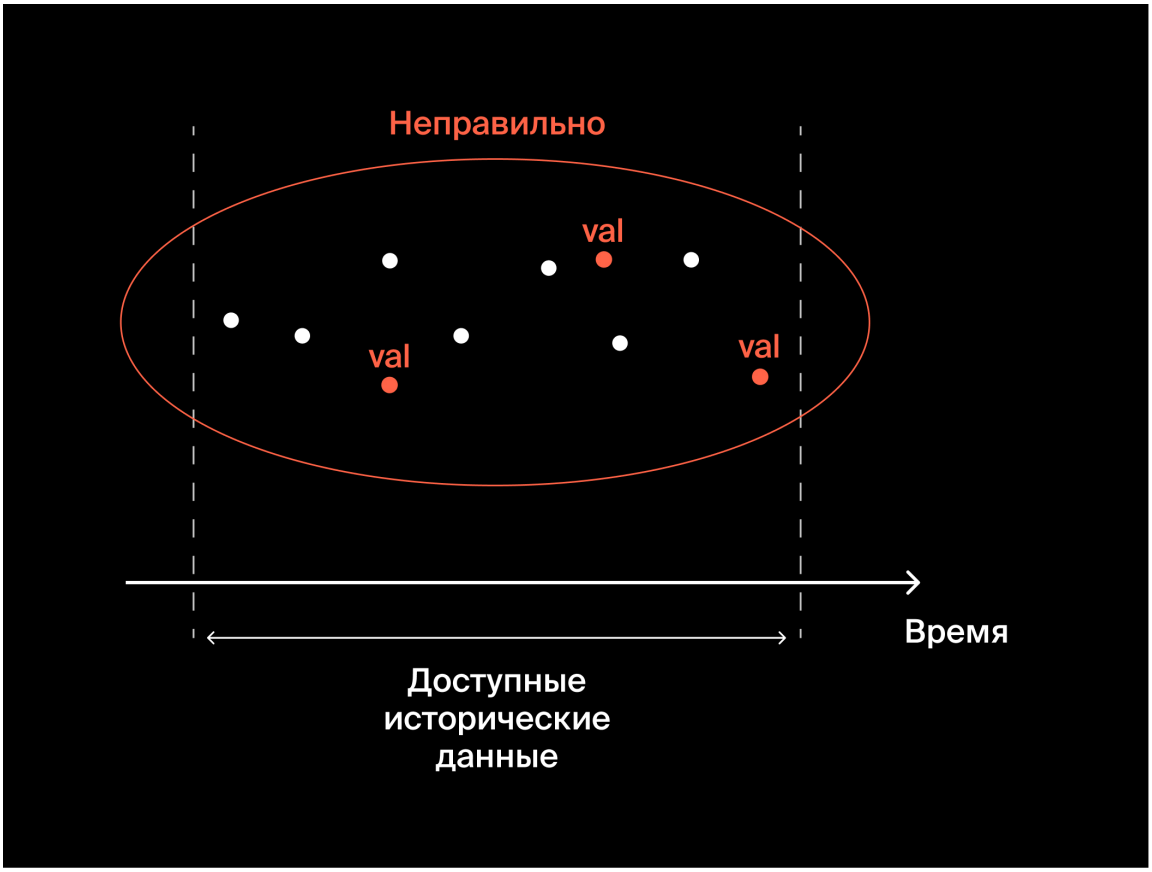
Случайное разбиение применяется, когда вы прогнозируете НЕ временные ряды, т. е. когда вы пренебрегаете влиянием соседних наблюдений друг на друга.

Временное распределение нужно, когда вы прогнозируете значение целевой переменной для последовательных во времени наблюдений. В таких задачах для каждого наблюдения мы часто прогнозируем значение целевой переменной «через N месяцев» — например, «через месяц» (как в нашей задаче), «через 2 месяца» и другие. А в качестве признаков

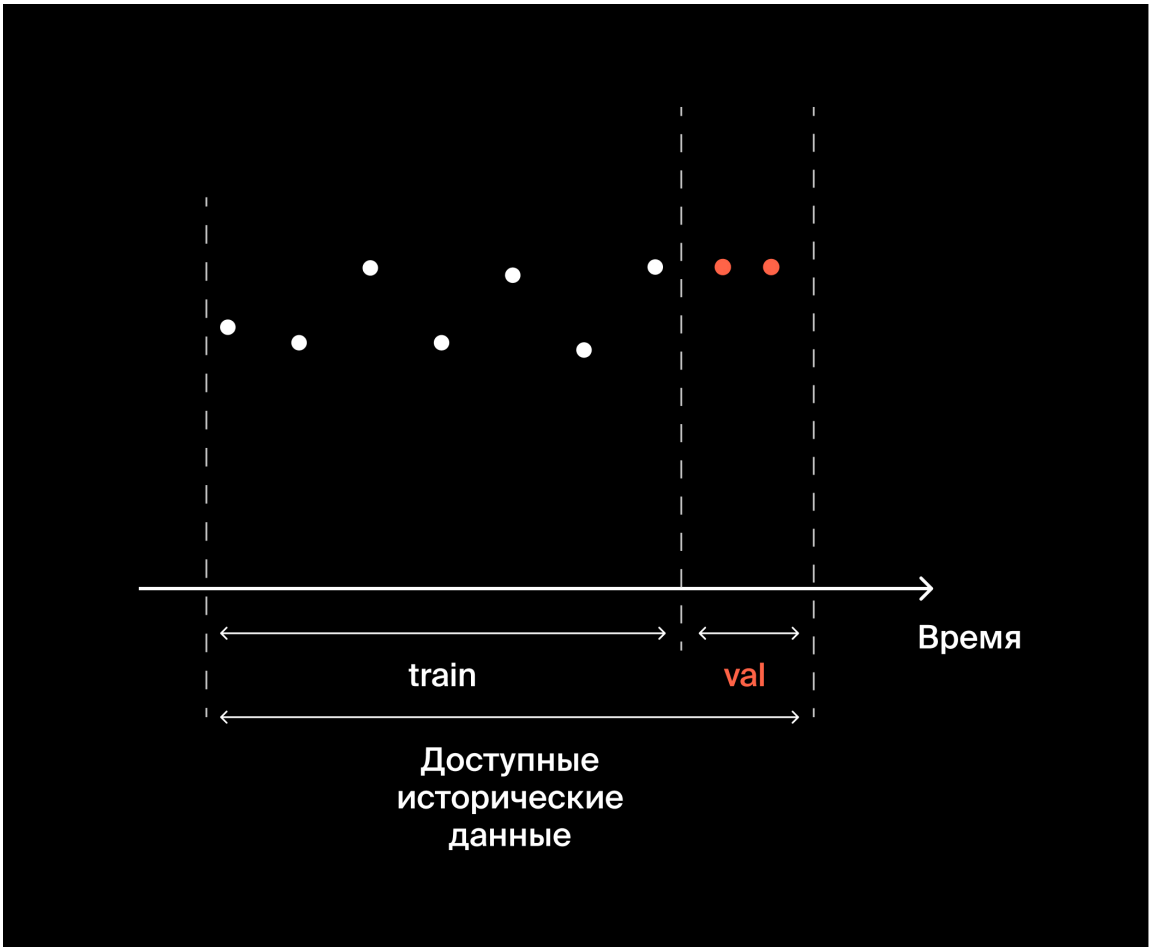
берут текущие значения переменных и другие признаки, сформированные с учётом времени. Причём даже для самой целевой переменной:

- лаги из прошлого. Например, *значения признаков и целевой переменной 1, 3, 5, 10, 30 и любое другое количество месяцев назад* (или минут, смотря по гранулярности данных, т. е. по принятой частоте временных интервалов).
- значения агрегирующих функций (сумма, среднее, стандартное отклонение, медиана) для признаков и целевой переменной за скользящие временные окна. Например, «*сумма значений за последние 3 месяца», «сумма значений за последние 5 месяцев», «среднее значение за последние 3 месяца», «среднее значение за последние 5 месяцев»*.

  Каждое наблюдение — точка во времени — тянет за собой информацию из прошлого. В этом случае случайное разбиение некорректно:



Корректно разбивать данные по времени:



Обучаясь на данных для временных рядов и моделируя валидацией реально отложенную выборку «из будущего» для алгоритма «из прошлого», правильнее использовать **временной сплит (time split или time-based split).** Это значит, что мы откладываем для валидации наблюдения с конца доступного исторического периода, а не случайно выбранные точки во времени. Для такого разбиения соотношение train/val чаще всего берут такое же, как для случайного разбиения — 80:20 или 70:30:

Выбор метрик

Для оценки качества ваших моделей и выбора из них лучшей вы определяете наиболее подходящие метрики, которые отражают суть вашей задачи. Два важных момента, которые мы ещё не разбирали:

1. после выбора приоритетной метрики можно так настроить процесс обучения, чтобы он эту метрику оптимизировал. Скажем, решая

задачу регрессии и используя *RandomForestRegressor()*, задаёте интересующую вас метрику напрямую при оптимизации обучения алгоритма:

```
model = RandomForestRegressor(criterion='mae')
```

Так алгоритм не столь чувствителен к случаям, когда модель сильно ошибается на отдельных наблюдениях — будете оптимизировать обучение, сводя к минимуму модуль ошибки. А если вы зададите модель таким образом:

```
model = RandomForestRegressor(criterion='mse')
```

модель за ошибки будет «оштрафована» сильнее, так как разница реального и спрогнозированного значений здесь возводится в квадрат, а не просто считается по модулю.

Здесь это два единственно возможных значения параметра `criterion`, но у других алгоритмов их бывает больше. Иногда, наоборот, среди доступных для настройки метрик нет вашей целевой и приходится выбирать что-то близкое к ней. Например, вы решаете задачу бинарной классификации и выбрали базовой метрикой *precision*, исходя из целей бизнеса. Алгоритм обучается реализацией градиентного бустинга из библиотеки *xgboost XGBClassifier()*. У этого алгоритма есть параметр `eval_metric`, который позволяет настраивать обучение под оптимизацию выбранной метрики. Однако среди возможных значений нет *precision*. В этом случае можно выбрать значение *auc*, которое будет стремиться улучшить ваш классификатор в общем, а *precision* вы уже будете регулировать за счёт «порога».

Вот почему всегда полезно читать документацию с описанием алгоритма и выяснять, каким способом его можно оптимизировать.

2. Для некоторых бизнес-задач приходится придумывать собственные метрики оценки качества модели на основе имеющихся. Простой и распространённый пример — «доля ложных срабатываний». По сути это отношение FP (ложных положительных ответов) к размеру вашей выборки. Если каждое срабатывание чего-то вам стоит, ваша цель — чтобы модель как можно реже делала это ошибочно. Как и

precision_score, эта метрика характеризует точность ваших срабатываний, и через эту точность вычисляется:

```
false_positive_rate = 1 - precision_score(y_true, y_pred)
```

Важность признаков

Вы выбрали лучшую модель, она работает, даже делает прогнозы. Чтобы от них был толк, в них должны поверить те сотрудники, которые эти прогнозы заказывали. И самое главное — вы сами. А чтобы поверили вы сами, вам нужно понимать, почему ваша модель работает. Здесь на первый план выходит такое её свойство, как «интерпретируемость».

Чаще всего аналитик готовит пачку данных, отправляет в черный ящик (*black box*) какого-нибудь алгоритма и на выходе получает прогноз. Однако нужно хотя бы общее понимание не только того, «что» сказала модель, а того, «как» и «почему» она это сделала. Какие признаки она посчитала важными при расчёте конкретного прогноза? Это очень важно для погружения в суть моделируемого процесса и вообще в оценке адекватности модели. Поэтому особое внимание следует уделить анализу её **важности признаков (или *feature importance*).**

Проще всего с линейными моделями. Обучение состоит в подборе коэффициентов для каждого признака. Так как эти признаки стандартизированы, значение коэффициента для каждого показывает его важность. Допустим, в ходе обучения модель подобрала следующие оптимальные коэффициенты для уравнения:

$$y = 132 + 37 * x_1 + 105 * x_2 + 0.3 * x_3 - 79 * x_4$$

Что будет, если мы изменим значение третьего признака x_3 с 0 на 1? В принципе, ничего особо не поменяется — ответ изменится на 0.3, что довольно мало. То есть третий признак не особо влияет на прогноз модели — его важность не очень высока. Со вторым признаком противоположная история: даже небольшое его изменение (снова с 0 на 1, например) может сильно повлиять на результат — то есть этот признак, наоборот, для модели очень важен. Коэффициент четвёртого признака равен -79. Он отрицательный, то есть если мы будем увеличивать

значение данного признака, прогнозная величина будет снижаться. А модуль этого коэффициента довольно высок, и меняться она будет заметно. Поэтому 4-й признак имеет высокую важность для прогноза.

Коэффициенты линейной регрессии хранятся в атрибуте обученной модели *.coef_* :

```
feature_weights = model.coef_
```

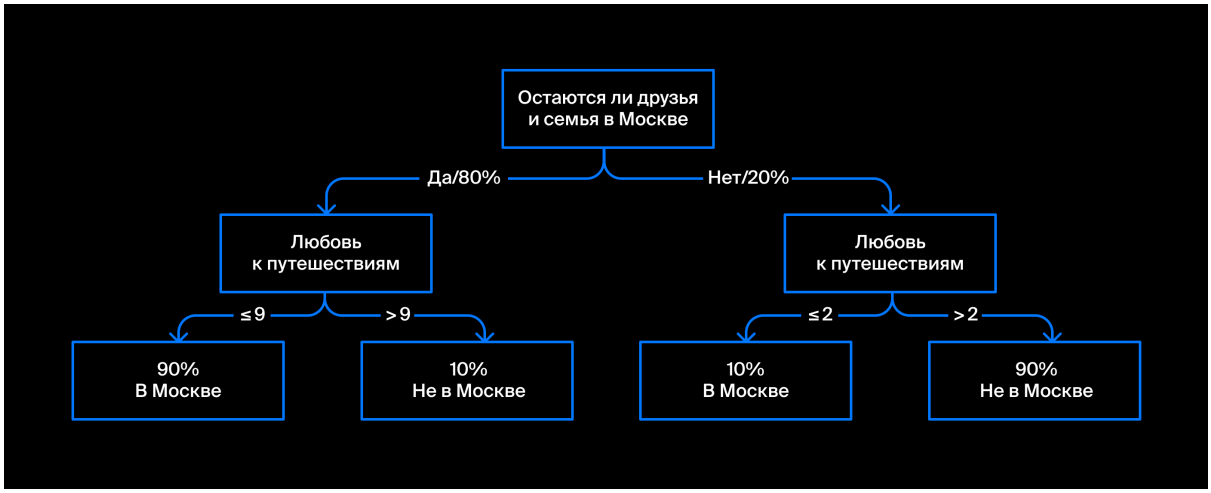
Так вы сохраните коэффициенты для всех признаков в переменной `feature_weights` («веса признаков»).

Чтобы вывести нулевой коэффициент (значение прогноза, когда значения всех признаков равны 0), используется атрибут *.intercept_:*

```
weight_0 = model.intercept_
```

Хорошо. С линейными моделями разобрались — смотрим на модули коэффициентов. А что же деревья? Если дерево одно, чаще используется тот параметр, который сильнее всего предопределяет прогнозы в последующих узлах дерева. Важность признаков определяется не коэффициентом, а рангом.

Тонкий момент при работе с деревьями — чем ближе признак к основанию дерева, тем выше его важность и способность влиять на результат прогноза.



Важность признаков для дерева (одинаково для алгоритмов регрессии и классификации) хранится в атрибуте обученной модели *.feature_importances_* :

```
importances = model.feature_importances_
```

Если определение важности признаков для дерева уже не самая очевидная задача, то для ансамблей моделей (градиентного бустинга и случайного леса) это исследовательская область, в рамках которой разрабатываются специальные алгоритмы. Однако на практике вам, возможно, не придётся настолько глубоко погружаться в теоретические аспекты вычисления важности. Для реализаций случайного леса и градиентного бустинга в sklearn важность признаков также хранится в атрибуте *.feature_importances_*