KARPOV.COURSES >>>
КОНСПЕКТ

Конспект > 8 урок > Полезные приемы при работе с данными

> Работа с пропущенными значениями

Зачем нужно заполнять пропущенные значения и почему нельзя просто удалить строчки с пропусками?

Для успешного обучения модели необходимо, чтобы она просмотрела как можно больше зависимостей. В свою очередь, удаление строк с пропущенными значениями приведет к снижению объема данных, на котором может обучиться модель. Например, при удалении всех строк с пустыми значениями в таблице внизу, она стала бы пустой.

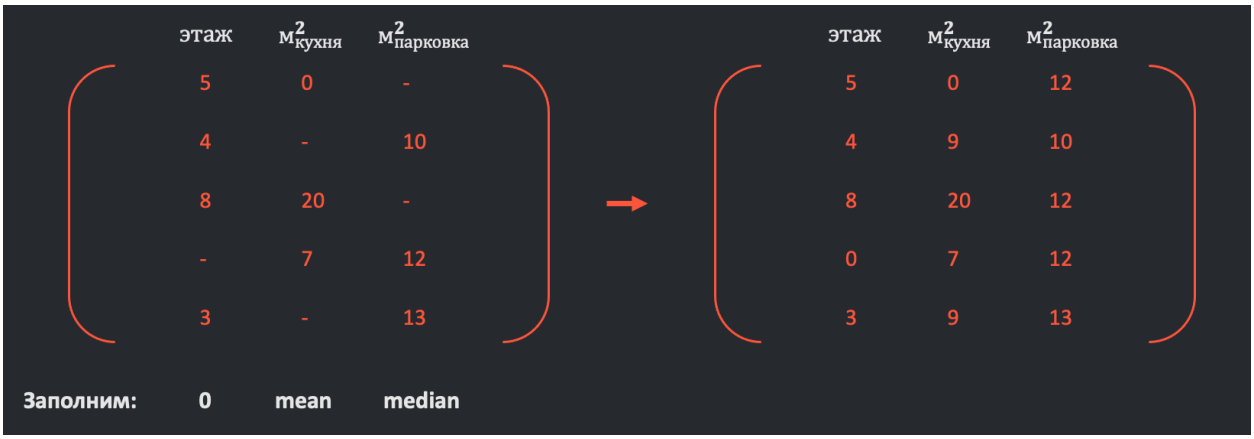
Существует ряд методов, которые применяются в работе с пропущенными значениями.

Например, можно:

- Заполнить все пропуски константой;
- Заполнить пропуски предыдущим значением;
- Для категорий – самым популярным классом;
- Или для категорий – ввести новый класс для всех пропусков;
- Посмотреть на то, какие значения встречаются у похожих объектов.

> Приемы для работы с пропущенными значениями

Одним из способов для заполнения пропущенных значений является заполнение константой. В качестве константы можно использовать нули, среднее колонки или медиану колонки.



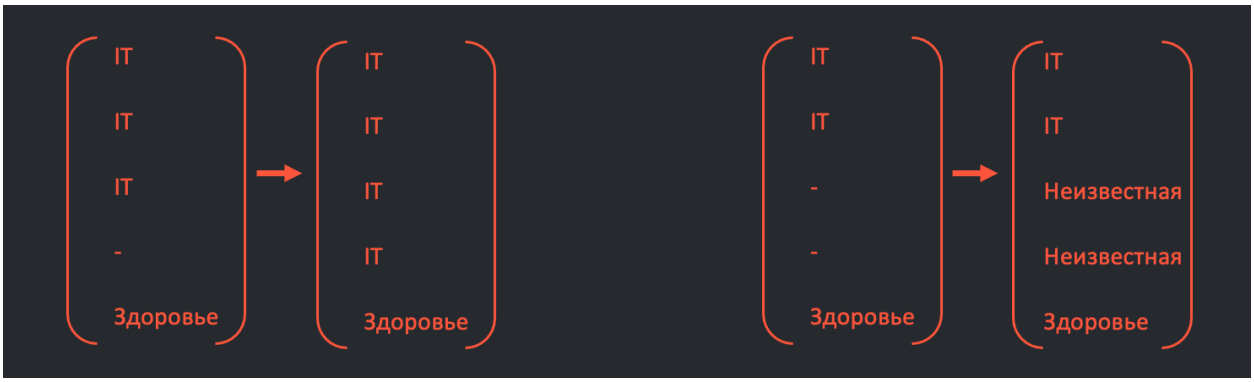

```
# Заполнить пропуски средним
mean = data['Цель в долларах'].mean()
data['Цель в долларах'].fillna(mean)
```

Заполнить пропуски предыдущим значением

Можно заполнять пропуски предыдущим значением в колонке.

Заполнить пропуски в категориальных признаках

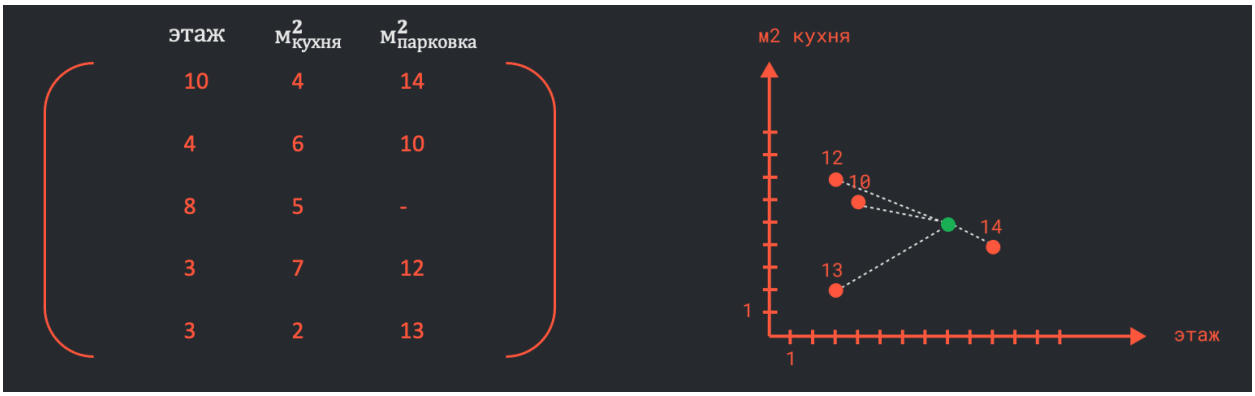
При работе с категориальными признаками можно заполнить пропуски самым популярным классом или же ввести новую категорию и заполнить ей пропуски.




```
# Заполнить пропуски самым популярным классом

popular_category = data['Главная категория'].value_counts().index[0]
data['Главная категория'] = data['Главная категория'].fillna(popular_category)

# Заполнить пропуски новой категорией

data['Валюта'] = data['Валюта'].fillna('Неизвестная валюта')
```

Посмотреть на похожие объекты

Можно рассмотреть каждую строку как точку в системе координат, затем сравнить расстояния между всеми точками. Логично предположить, что близкие объекты будут похожи друг на друга, поэтому можно заполнить пропуски аналогичными значениями из самых ближайших объектов.



```
# Заполнить пропуски, ориентируясь на похожие объекты

grouped_means = data.groupby('Главная категория')['Цель в долларах'].transform("mean")
data['Цель в долларах'] = data['Цель в долларах'].fillna(grouped_means)
```

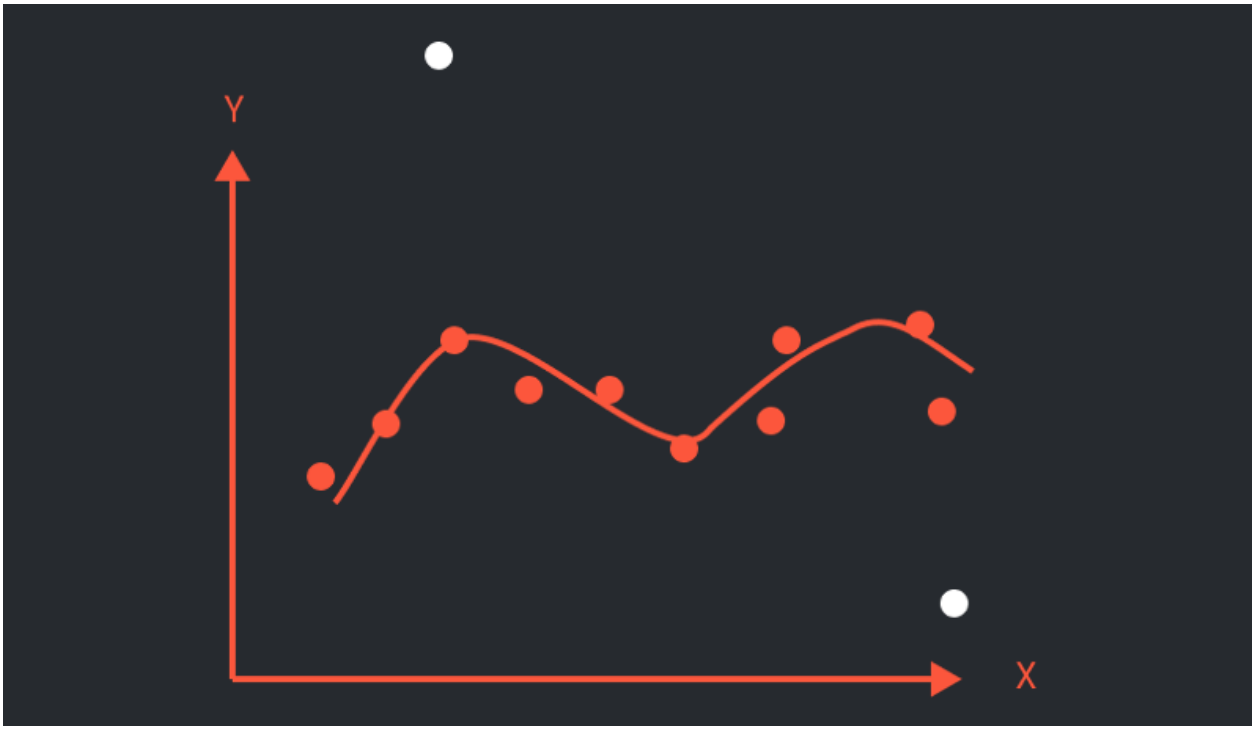
Резюме

- Пропуски в данных могут быть как в таргете, так и в признаках;
- Есть ряд приемов, которые помогают заполнять эти пропуски;
- Приемы отличаются между собой по простоте реализации;
- То, какой прием лучше применить, будет зависеть от конкретной ситуации.

> Работа с выбросами

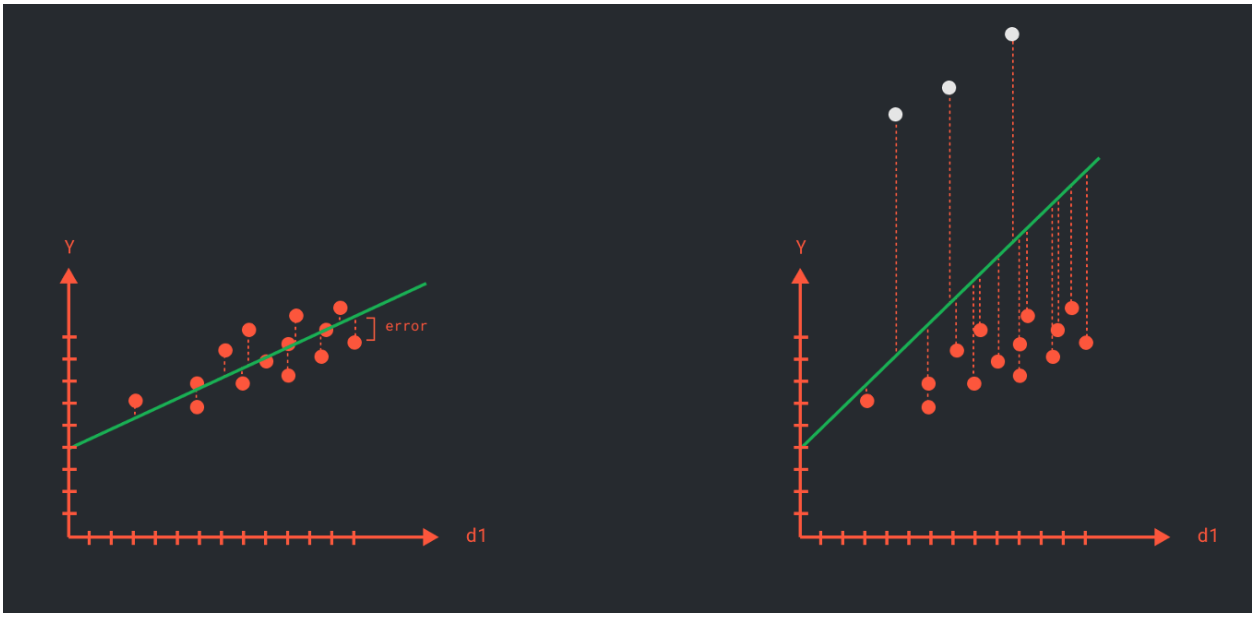
Пусть известно, что есть зависимость между признаками X и Y. Тогда выбросами можно назвать те точки, которые выбиваются из общего распределения на величину большую, чем шум.

Ниже на графике выбросами являются белые точки:



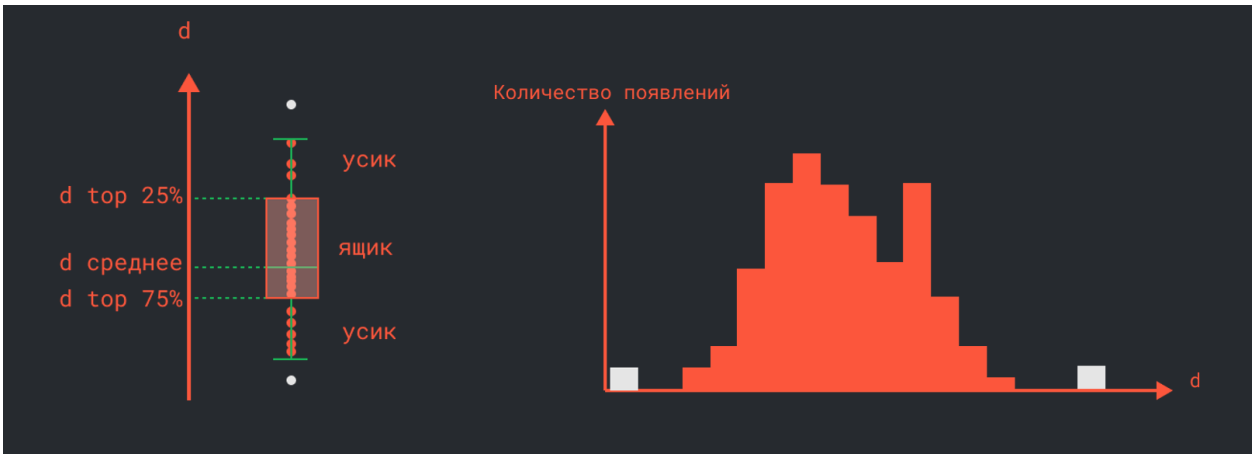
Как выбросы влияют на модель?

Выбросы будут стягивать линию регрессии на себя, тем самым ухудшая функциональность модели (её обобщающую способность), а также увеличивая среднеквадратические ошибки.



Как можно идентифицировать выбросы?

Можно находить выбросы, используя график боксплот — ящик с усами, и убирать значения, выходящие за пределы усов графика. Также можно использовать гистограмму распределения значений и убирать значения, которые находятся в «хвостах» гистограммы (редкие значения).



Как чистить данные от выбросов на примере боксплота?

1. Посчитать длину ящика:

$$r = d25\% - d75\%$$

2. Посчитать кончики усов:

$$d_{25\%} + 1.5 \cdot r$$

$$d_{75\%} - 1.5 \cdot r$$

3. Вывести критерий отсечения выбросов:

$$di \in [d_{75\%} - 1.5 \cdot r; d_{25\%} + 1.5 \cdot r]$$

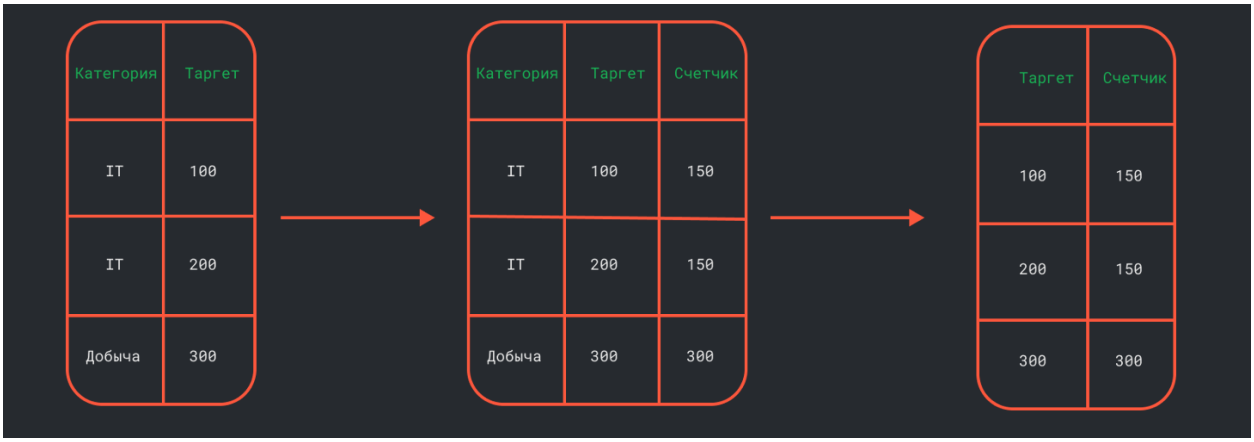
Резюме

- В переменных могут содержаться выбросы;
- Они усложняют построение адекватной модели;
- Они сильно портят метрики;
- Чтобы их обнаружить, можно исследовать ящики с усами и гистограммы;
- Есть еще более продвинутые методы, как отличить выброс от невыброса;
- Так или иначе, если хотя бы большая часть выбросов будет найдена, станет гораздо лучше.

> Advanced счетчики

Для начала вспомним процедуру создания обычного счетчика.

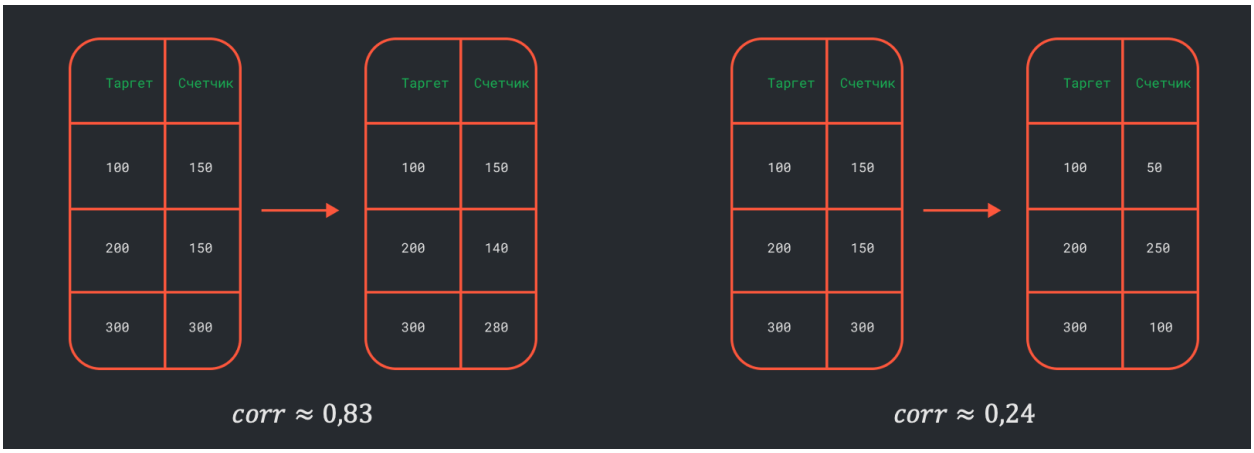
Пусть имеется какой-либо категориальный признак, который нужно раскодировать. Для этого нужно посчитать среднее значение таргета для каждой категории.



Такая кодировка может привести к переобучению, так как, считая данный признак, приходится подглядывать в таргет, и из-за этого счетчики коррелируют с ответами.

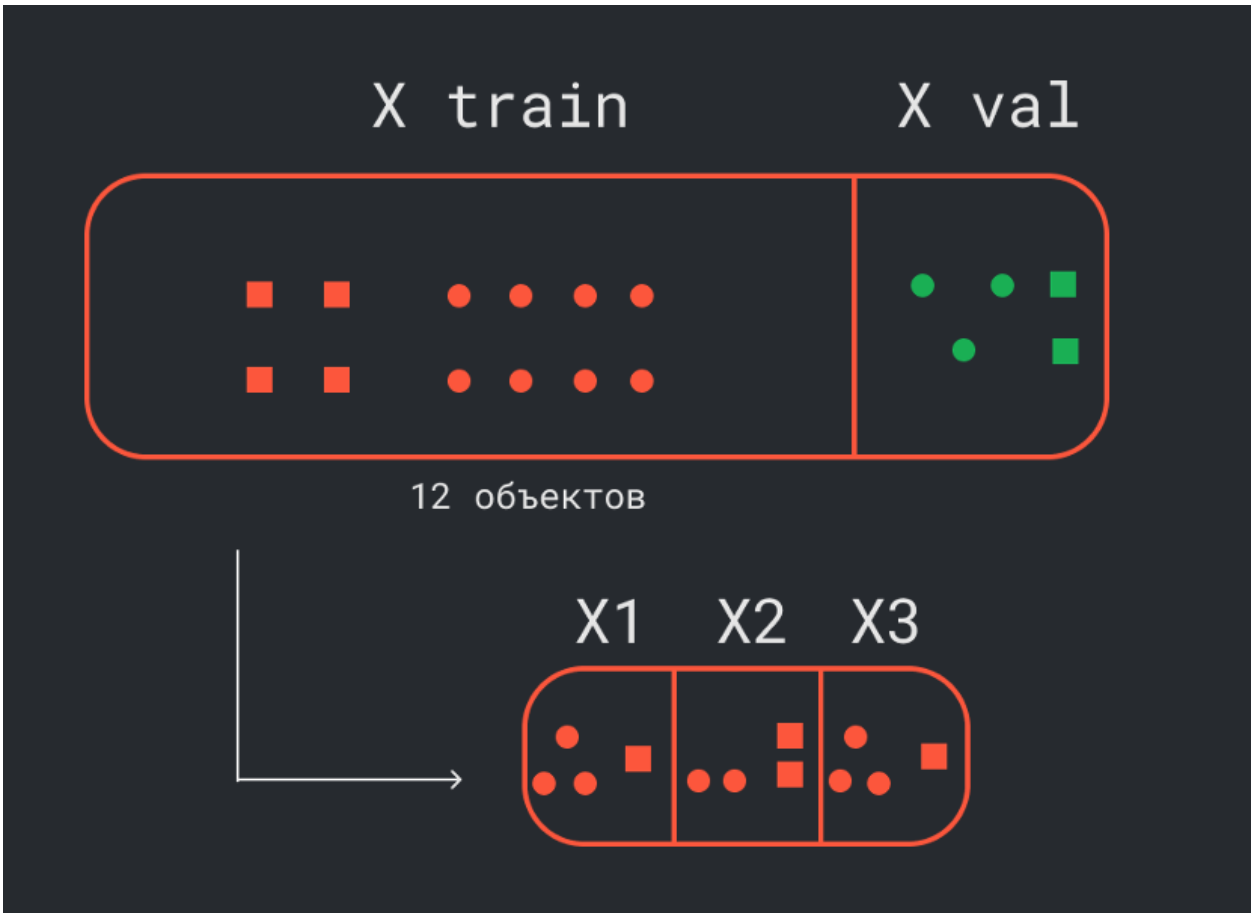
Как избежать переобучения?

Можно, например, добавить шум к данным. Однако, как видно на картинке внизу, добавление шума приводит к снижению корреляции и качества модели, а этого хотелось бы избежать.



Существует и другой способ избегания переобучения — кросс-валидация модели.

Обычно выборка делится на тренировочную и валидационную. Для кросс-валидации нужно дополнительно поделить тренировочную выборку на несколько подвыборок. Затем для категорий из блока X1 подсчитать счетчики на основе таргетов этих категорий в блоках X2, X3. То же самое нужно сделать для других блоков — для X2 нужно использовать блоки X1 и X3, а для X3 — блоки X1 и X2. Таким образом мы дополнительно тренируем и валидируем модель на искусственно созданной валидационной выборке.



Резюме

- Счетчики часто ведут к переобучению моделей;
- Чтобы этого избежать при их подсчете можно воспользоваться одним из двух приемов:
- (1) Добавить в полученные значения случайный шум;
- (2) При формировании счетчиков считать их по другим объектам.

> Выделение признаков из текста

Зачем нужно выделять признаки из текста?

Наименования объектов обычно уникальны. Они воспринимаются как одно целое — id. С
другой стороны, можно попробовать погрузиться в смысл описания объекта и перевести текстовое описание объекта в мир чисел и матриц. Там может крыться что-то очень важное, что поможет предсказывать таргет.

Например, в таблице внизу есть явная зависимость бюджета проекта от его описания. Как же понять, какие слова влияют на успех?

Наименование проекта	Признаки	Сколько $$ соберет?
Запись скучного ROCK альбома	...	200
Много плюшек инвесторам! Приходите!!	...	2000
Всем подарки! Без СМС и регистрации!	...	3500
Фэнтези-роман про бедного фермера	...	300
Кофе в переходах метро	...	300
Первым инвесторам - большие скидки!	...	1500

Для этого существует принцип "bag of words" — для каждого слова подсчитываем его встречаемость в тексте.

Наименование проекта	Инвестору	-	!	Дарим	...
Инвестору - скидка	1	1	0	0	...
Дарим каждому инвестору подарок!	1	0	1	1	...
Дарим каждому инвестору бонус!	1	0	1	1	...

Преимущества

- Простой в применении;
- Не требует обработки текста.

Недостатки

- Может порождать огромные матрицы;
- Не учитываем общий контекст задачи и среднюю частоту всех слов;
- Например, если мы прогнозируем цены на овощи, то очевидно, что в описании к объекту почти всегда будет слово «овощ», то есть необходимо погружаться в контекст.

> TF-IDF

Для избежания упомянутых проблем нужен метод для более умной векторизации описания объектов.

С этим справляется трансформация текста методом TF-IDF - *term frequency-inverse document frequency*. Он позволяет выделить самые важные слова в каждом конкретном описании, учитывая общий контекст задачи и то, что написано в других объектах.

Данный метод состоит из двух компонент: TF и IDF.

Как подсчитывается TF?

Здесь нужно подсчитать, сколько раз слово входит в описание, затем поделить это число на общее количество слов.

$tf(t, d) = \frac{n_t}{(\sum_i n_i)}$, где n_t — это количество вхождений слова t в описание d.

Таким образом можно узнать, насколько данное слово важно для конкретного описания. Чем чаще слово встречается в описании, тем больше у этого слова в данном описании TF.

Как подсчитывается IDF?

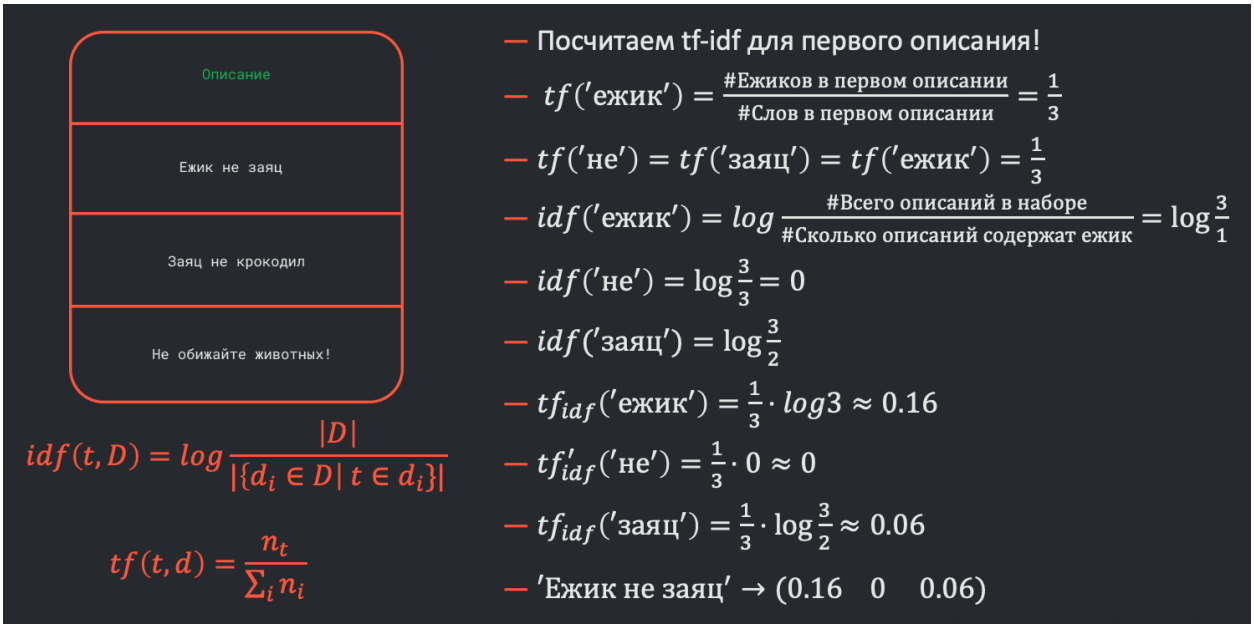
Здесь нужно взять логарифм частного количества всех описаний и количества описаний, которые содержат данное слово.

$idf(t, D) = log(\frac{|D|}{|\{di \in D | t \in di\}|})$, где $|\ D\ |$ – это количество описаний, а знаменатель – это количество описаний с словом t.

Таким образом можно узнать, насколько данное слово является общим для всех описаний. Чем чаще слово встречается в описании других объектов, тем меньше у него IDF.

Финальной метрикой является произведение TF и IDF. Большие значения TF-IDF будут иметь слова, которые часто встречаются в конкретном документе, но при этом оказываются редкими в других.

Ниже представлен пример вычисления TF-IDF.



Как использовать данную метрику?

В итоге мы получаем некоторый множественный признак, поверх которого можно сделать агрегацию, например, посчитать максимум или среднее.

Преимущества TF-IDF

- Более компактный, чем bag of words;
- Учитывает важность слов.

Недостатки TF-IDF

- Получаем оценку важности каждого слова через частности, но не погружаемся в контекст;
- Как и bag of words, у данного метода есть проблемы с однокоренными словами, падежами, склонениями и так далее.

Реализация TF-IDF в sklearn

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Зафиттим наши данные в TfidfVectorizer
data = data.dropna()
tfidf = TfidfVectorizer()
tfidf.fit(data['Название'])

### Посмотрим как выглядит наш первый документ (первое описание)
first_document = data['Название'][0]

### Векторизуем данное описание через tf-idf
tfidf.transform([first_document])
tfidf.transform([first_document]).todense()
tfidf.get_feature_names()

### Посмотрим на содержимое этого вектора
df = pd.DataFrame(tfidf.transform([first_document]).T.todense(),
                  index=tfidf.get_feature_names(),
                  columns=['tfidf'])
```

> Лемматизация и стемминг

Для того, чтобы привести все слова к одной форме, используются два подхода — лемматизация и стемминг.

Лемматизация – приводит все слова к единой форме, например, к единственному числу, мужскому роду и именительному падежу.

Стемминг — находит основу слова.