4 метода векторизации текстов

Автор [Роман Котюбеев](https://python-school.ru/author/favorshlitex/)

Обработка естественного языка или [NLP](https://python-school.ru/wiki/nlp/) (Natural Language Processing) занимается применением алгоритмов Machine Learning для текстовых данных. Как правило, модели машинного обучения работают с числами. В этой статье поговорим о 4-х наиболее применяемых методах для перевода текстов в числовые тензоры.

Основные термины NLP: корпус, документ, токен, словарь

Сначала текст разбивается на текстовые единицы (**токены**), например, символы, слова, словосочетания, предложения, абзацы и т.д. Чаще всего разбивают на слова. Токены образуют **словарь**, который может быть отсортирован по алфавиту.
Также в [NLP](https://python-school.ru/wiki/nlp/) применяются термины «**документ**» и «**корпус**». Документ – это совокупность токенов, которые принадлежат одной смысловой единице. В качестве документа может выступать предложение, комментарий или пост пользователя. Корпус – это генеральная совокупность всех документов.
Рассмотрим пример. Допустим имеется два предложения: “Пес сел на пень”, “Кот сел на ель”. Выберем в качестве токенов слова, тогда получится следующий словарь:

{Пес, Кот, ель, на, сел, пень} # Словарь

и два документа, которые составляют корпус:

[Пес, сел, на, пень] # Первый документ

[Кот, сел, на, ель] # Второй документ

[[Пес, сел, на, пень], [Кот, сел, на, ель]] # Корпус

В последующих методах кодирования слов мы также будем использовать два этих предложения в качестве примера.

1. Прямое кодирование

Прямое кодирование (one-hot encoding) считается самым простым способом преобразования токенов в тензоры и выполняется следующим образом:

1. каждый токен представляет бинарный вектор (значения 0 или 1);
2. единица ставится тому элементу, который соответствует номеру токена в словаре.

С нашими предложениями это выглядит так:

{Пес, Кот, ель, на, сел, пень} # Словарь

# Первый документ

[[1, 0, 0, 0, 0, 0]  # Пес

[0, 0, 0, 0, 0, 0]  # Кот (нет в предложении)

[0, 0, 0, 0, 0, 0]  # ель (нет в предложении)

[0, 0, 0, 1, 0, 0]  # на

[0, 0, 0, 0, 1, 0]  # сел

[0, 0, 0, 0, 0, 1]] # пень

# Второй документ

[[0, 0, 0, 0, 0, 0]  # Пес (нет в предложении)

[0, 1, 0, 0, 0, 0]  # Кот

[0, 0, 1, 0, 0, 0]  # ель

[0, 0, 0, 1, 0, 0]  # на

[0, 0, 0, 0, 1, 0]  # сел

[0, 0, 0, 0, 0, 0]] # пень (нет в предложении)

Проблемой прямого кодирования является размерность. Каждое предложение состоит всего из 4 слов, но в итоге получилась большая матрица для каждого документа. Количество строк регулируется словарем, поэтому чем больше слов в словаре, тем больше будет матрица.

2. Bag of words

В отличие от прямого кодирования, мешок слов ([Bag of words](https://python-school.ru/wiki/bag-of-words/%22%20%5Ct%20%22_blank)) выделяет вектору весь документ, и каждый элемент кодируется 1 по порядку следования слов в словаре:

{Пес, Кот, ель, на, сел, пень} # Словарь

# Корпус:

[[1, 0, 0, 1, 1, 1]  # Первый документ

 [0, 1, 1, 1, 1, 0]] # Второй документ

[Bag of words](https://python-school.ru/wiki/bag-of-words/) решает проблему размерности по одной оси. Количество строк определяется количеством документов. Однако, этот метод не учитывает важность того или иного токена, ведь одно слово может повторятся по несколько раз. В этом случае пригодится альтернативный способ, рассмотренный далее.

3. TF-IDF

[TF-IDF](https://python-school.ru/wiki/tf-idf/) состоит из двух компонентов: Term Frequency (частотность слова в документе) и Inverse Document Frequency (инверсия частоты документа). Они считаются следующим образом:

  ![\[ TF_{token_i}=\frac{n_i}{N_i}, \]]()

  ![\[ IDF_{token}=\log{\frac{p}{P}}, \]]()

где  — сколько раз встречается токен в -ом документе,
 — общее количество токенов в -ом документе,
 — количество документов, в которых встречается токен,
 — общее количеств документов.
В конечном счете, [TF-IDF](https://python-school.ru/wiki/tf-idf/) – это произведение TF на IDF:

  ![\[ TF\textrm{-}IDF = TF\times IDF \]]()

Стоит отметить, что TF считается для токенов документа, тогда как IDF – токенов всего корпуса. Итак, у нас имеется 2 документа, в каждом из которых оказалось по 4 слова. В этом случае вычисления будут следующими:

Вычисления [TF-IDF](https://python-school.ru/wiki/tf-idf/)

В результате получили для 1-го документа такие важные слова, как “Пес” и “пень”, для 2-го — “Кот” и “ель”. В [TF-IDF](https://python-school.ru/wiki/tf-idf/) редкие слова и слова, которые встречаются во всех документах, несут мало информации. Кроме того, IDF можно считать и другими способами, например, в Python-библиотеке [Scikit-learn](https://python-school.ru/wiki/scikit-learn/%22%20%5Ct%20%22_blank) этот параметр гибко [регулируется](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html).

4. Word Embeddings

Все вышерассмотренные [NLP](https://python-school.ru/wiki/nlp/)-методы отличаются следующими недостатками:

* не зависят от контекста – например, оба анализируемых предложения отражают примерно одно и то же: “что-то куда-то село”;
* не учитывают порядок слов в предложении;
* обладают высокой размерностью в случае большого словаря, что может снизить производительность модели глубокого обучения ([Deep Learning](https://python-school.ru/wiki/deep-learning/%22%20%5Ct%20%22_blank)).

На практике все чаще используется word embeddings – векторное представление слов. Векторы можно складывать, вычитать, сравнивать. Например, можно ли сложить слова “Король” и “Женщина”? Можно предположить, что будет “Королева”. А можно ли сравнить близость слов “мужчина”, “мальчик”, “девочка”? Напрашивается, что “мужчина” и “мальчик” стоят ближе друг к другу. На изображении показано, как это выглядит графически:

Векторное пространство слов

В этом примере на одной стороне животные, на другой люди. Со стороны животных возможен переход от домашних к диким или от псовых к кошачьим. Со стороны людей возможен переход по полу и возрасту. Именно так и работают word embeddings, представляя слова (токены) в векторы.
Самой распространенной реализацией векторного представления слов является **Word2vec**. Архитектура Word2vec подразделяется на два вида — Skip-gram и Continuous Bag of Words (CBOW).
Skip-gram получает на вход одно слово и предсказывает подходящий контекст. Например, предсказываем контекст к слову “Пес”, он может выглядеть так:

[сел]

[сел на]

[пень]

[сел на пень]

...

В свою очередь, CBOW пытается угадать слово, исходя из контекста. Например, следующее слово в предложении “Кот сел на […]” может быть следующим:

[ель]

[стул]

[пень]

...

Контекст для word embeddings является очень важным. Юридические документы отличаются от комментариев в социальных сетях, поэтому и результат может быть разным.