

# ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ДАННЫХ ДЛЯ МАШИННОГО ОБУЧЕНИЯ

Цель предварительной обработки данных — сделать исходные данные пригодными для машинного обучения. Сюда входят векторизация, нормализация, обработка недостающих значений и извлечение признаков.

## ВЕКТОРИЗАЦИЯ

Мешок слов (bag of words) - упрощенное представление текста, которое показывает, какие слова встретились в тексте, но при этом не учитывает их порядок.

Представление мешка слов — это таблица с числами, в которой столбцы таблицы — уникальные слова, а строки — документы коллекции. В ячейках таблицы находится число вхождений слова в документ. Значит, в каждой строке получится набор чисел (он же вектор), характеризующий состав документа.

Пусть у нас есть два текста: «Это были лучшие времена» и «Это было худшее время» (*ниже пример на английском, потому что библиотека работает с английским*). В обоих предложениях встречается суммарно 5 различных слов, если привести к начальным формам: «Это», «Быть», «Лучший», «Худший», «Время». Это будет наш словарь.

Выделим встреченные слова из словаря в текстах: в первом встретились [«Это», «Быть», «Лучший», «Время»], а во втором — [«Это», «Быть», «Худший», «Время»].

Векторное представление мешка слов для первого текста будет [1 1 1 0 1], где нолик стоит на месте элемента «Худший», так как оно не встретилось в нем, а для второго — [1 1 0 1 1], где нолик на месте слова «Лучший». Так мы перешли к упрощенному машиночитаемому представлению двух текстов.

```
import numpy as np
```

```
s1 = 'It was a good time'
```

```
s2 = 'It was a worst time'
```

```
s3 = [s1, s2]
```

```
words = list(set((s1+' '+s2).split()))
```

```
print(words)
```

```
results = np.zeros((len(s3), len(words)))
```

```
for i, sequence in enumerate(s3):  
    for j in sequence.split():  
        results[i, words.index(j)] = 1.  
  
print(results)
```

### **Задание 1.**

Данные - текстовые отзывы на фильмы, разделенные на негативные и позитивные. Что мы собираемся с ними сделать:

1. выделить отдельные слова, привести их к единому виду (убрав словоформы);
2. построить вектор из всех слов, а потом для каждого сообщения заполнить вектор такой же длины нулями или единицами в зависимости от того, было ли слово в тексте;
3. объединить все, разделить данные на обучающую и тестовую выборки;
4. создать классификатор и обучить его.

В папке с лабораторной скачайте файл data.zip.

В цикле считайте из папки neg txt'шники в list строк. В тот же list добавьте строки из папки pos. Параллельно создайте list со значениями '0' для негативных отзывов и '1' – для положительных.

Выполнить прямое кодирование списков слов в векторы нулей и единиц (в numpy массив). Это может означать, например, преобразование последовательности [good, day] в 10 000-мерный вектор, все элементы которого содержат нули, кроме элементов с индексами 3 и 5 (номеров этих слов в словаре), которые содержат единицы. Затем их можно передать на вход модели машинного обучения.

## Нюансы использования

В реальных задачах все сложнее. Чтобы не мусорить в таблице, из текста убирают служебные слова. Слова приводят не обязательно к начальной форме (см. «лемматизация»), но иногда и обрезают, оставляя только грамматическую основу. (см. «стемминг»). Поэтому правильнее называть их уже не словами, а «токенами». Иногда столбцы обозначают не отдельные слова, а пары подряд идущих слов (биграммы) или тройки (триграммы).

Чаще всего в ячейках пишут не абсолютный показатель «слово встретилось 15 раз», а относительный показатель из статистики: он называется tf-idf и описывает важность слова для классификации текста.

**Стемминг** – это грубый эвристический процесс, который отрезает «лишнее» от корня слов, часто это приводит к потере словообразовательных суффиксов.

**Лемматизация** – это более тонкий процесс, который использует словарь и морфологический анализ, чтобы в итоге привести слово к его канонической форме – лемме.

После векторизации нужно разбить выборку на обучающую и тестовую.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(train_data_features,
emotion, test_size=0.2, random_state=0)
```

## Создать и обучить классификатор

```
from sklearn.naive_bayes import BernoulliNB
clf = BernoulliNB()
```

```
clf.fit()
```

Посмотреть, что выдает классификатор на тестовой выборке и сравнить с правильными значениями.

```
from sklearn import metrics  
metrics.accuracy_score()
```

## НОРМАЛИЗАЦИЯ

Чтобы упростить обучение сети, данные должны обладать следующими характеристиками:

- принимать небольшие значения — как правило, значения должны находиться в диапазоне 0–1;
- быть однородными — то есть все признаки должны принимать значения из примерно одного и того же диапазона.

Кроме того, может оказаться полезной следующая практика нормализации, хотя она не всегда необходима (например, мы не использовали нормализацию в примере классификации цифр):

- нормализация каждого признака независимо, чтобы его среднее значение было равно 0;
- нормализация каждого признака независимо, чтобы его стандартное отклонение было равно 1.

Это легко реализуется с применением массивов NumPy:

```
x -= x.mean(axis=0)
```

```
x /= x.std(axis=0) (Предполагается, что x — это двумерная матрица данных с формой (образцы,  
свойства))
```

Глубокое обучение на Python. Шолле Франсуа

От стр.111. 3.6. Предсказание цен на дома: пример регрессии

До стр.112. 3.6.2. Подготовка данных (*Конструирование сети – не нужно*)

После подготовки данных, обучить модель Ridge и вывести значение среднеквадратичной ошибки.

```
from sklearn.linear_model import Ridge
```

## ОБРАБОТКА НЕДОСТАЮЩИХ ЗНАЧЕНИЙ

Иногда в исходных данных могут отсутствовать некоторые значения. Так, в примере с предсказанием цен на дома первым признаком (столбец с индексом 0 в данных) был уровень преступности на душу населения. Как быть, если этот признак определен не во всех образцах? Если оставить все как есть, у нас будет недоставать значений в тренировочных или в контрольных данных.

Можно заменить недостающие входные значения нулями, при условии, что 0 не является осмысленным значением. Обратите внимание на то, что, если в контрольных данных имеются отсутствующие значения, но модель была обучена без них, ваша модель не будет распознавать отсутствующие значения! В этой ситуации следует искусственно сгенерировать тренировочные экземпляры с отсутствующими признаками: скопируйте несколько тренировочных образцов и отбросьте в них некоторые признаки, которые, как ожидается, не определены в контрольных данных.

## КОНСТРУИРОВАНИЕ ПРИЗНАКОВ

Конструирование признаков — это процесс использования ваших собственных знаний о данных и алгоритме машинного обучения, чтобы улучшить эффективность алгоритма применением predetermined преобразований к данным перед передачей их в модель. Во многих случаях

не следует ожидать, что модель машинного обучения сможет обучиться на полностью произвольных данных. Данные должны передаваться в модель в виде, облегчающем ее работу.

Рассмотрим простой пример. Допустим, нам нужно разработать модель, принимающую изображение циферблата часов со стрелками и возвращающую время (рис. 4.3).



Исходные данные: сетка с пикселями		
Более удачные признаки: координаты стрелок	$\{x1: 0.7,$ $y1: 0.7\}$ $\{x2: 0.5,$ $y2: 0.0\}$	$\{x1: 0.0,$ $y2: 1.0\}$ $\{x2: -0.38,$ $2: 0.32\}$
Еще более удачные признаки: углы отклонения стрелок	$\theta1: 45$ $\theta2: 0$	$\theta1: 90$ $\theta2: 140$

Рис. 4.3. Конструирование признаков для чтения времени с изображения циферблата часов

Если в качестве входных данных использовать пиксели исходных изображений, вы столкнетесь со сложной задачей машинного обучения. Для ее решения вам потребуется сконструировать сверточную сеть и потратить большой объем вычислительных ресурсов на ее обучение.

Однако, понимая задачу на высоком уровне (как человек определяет время по циферблату часов), можно сконструировать гораздо более удачные входные признаки для алгоритма машинного обучения: например, можно с легкостью написать пять строк кода на Python, которые последуют по

черным пикселям стрелок и вернут координаты  $(x, y)$  конца каждой стрелки. Тогда простой алгоритм машинного обучения сможет обучиться связывать эти координаты с соответствующим временем дня.

Можно пойти еще дальше: преобразовать координаты  $(x, y)$  в полярные координаты относительно центра изображения. В этом случае на вход будут подаваться углы отклонения стрелок. Полученные в результате признаки делают задачу настолько простой, что для ее решения не требуется применять методику машинного обучения; простой операции округления и поиска в словаре вполне достаточно, чтобы получить приближенное значение времени дня.

В этом суть конструирования признаков: упростить задачу и сделать возможным ее решение более простыми средствами. Обычно для этого требуется глубокое понимание задачи.

До глубокого обучения конструирование признаков играло важную роль, поскольку классические поверхностные алгоритмы не имели пространств гипотез, достаточно богатых, чтобы выявить полезные признаки самостоятельно. Форма данных, передаваемых алгоритму, имела решающее значение для успеха.

К счастью, современные технологии глубокого обучения в большинстве случаев избавляют от необходимости конструировать признаки, потому что нейронные сети способны автоматически извлекать полезные признаки из исходных данных. Означает ли это, что вы не должны беспокоиться о конструировании признаков при использовании глубоких нейронных сетей? Нет, и вот почему:

- Хорошие признаки позволяют решать задачи более элегантно и с меньшими затратами ресурсов. Например, было бы смешно решать проблему чтения показаний с циферблата часов с привлечением сверточной нейронной сети.
- Хорошие признаки позволяют решать задачи, имея намного меньший объем исходных данных. Способность моделей глубокого обучения

самостоятельно выделять признаки зависит от наличия большого объема исходных данных; если у вас всего несколько образцов, информационная ценность их признаков приобретает определяющее значение.