# Урок 6

# Текстовые данные и работа с ними

## 6.1. Работа с текстовыми данными

Эта неделя посвящена работе с текстами. Будут рассмотрены особенности текстовых данных, способы извлечения из них признаков, типичные постановки задач, а также несколько прикладных кейсов.

### 6.1.1. Примеры задач

Текстовые данные встречаются повсюду. В частности, можно сказать, что интернет — это огромная коллекция текстов различных форматов. С их помощью решают большое количество задач:

- предсказание по тексту записи в блоге её рейтинга. Успешность записи можно оценить ещё до публикации;
- определение эмоционального окраса комментария (положительный или отрицательный). Это может пригодиться при анализе отзывов о банке от клиентов, при обнаружении отрицательного отзыва появляется возможность быстро предпринять какие-либо действия и повысить лояльность клиента;
- определение тематики научной статьи при добавлении её в библиотеку. После этого можно либо автоматически отнести текст к выявленной категории, либо предложить автору варианты выбора;
- кластеризация новостей по сюжетам. Об одном и том же событии пишут разные новостные сайты, и имеет смысл группировать новости с одинаковым содержанием;
- поиск слов, похожих по смыслу на данное. Например, слова «стул» и «кресло» немного отличаются по смыслу, но всё-таки означают примерно одно и то же. Для компьютера оба слова — это набор символов, и с помощью простых алгоритмов нельзя определить, что это синонимы. О том, как делать такие выводы из данных, будет рассказано позднее.

Существуют и более сложные задачи на текстовых данных:

- выделить все упоминания имён в тексте. Это задача выделения новых сущностей;
- построение краткой аннотации текста;
- создание модели, отвечающей на вопросы. Она принимает на вход вопрос в произвольной форме и пытается найти на него ответ. При этом ответ строится по большому неструктурированному корпусу, например, википедии;
- генерация текста, похожего на заданный набор. Было бы интересно использовать, например, собрание сочинений Л.Н. Толстого и по нему построить модель, генерирующую тексты, похожие на рассказы автора. Уже сейчас эту задачу пытаются решать, и далее будет упомянуто, как это можно делать.

### 6.1.2. Текстовые признаки

Итак, задача — научиться использовать текстовые данные в алгоритмах машинного обучения: линейных моделях, решающих деревьях, нейронных сетях. Основная проблема заключается в том, что текст — это набор объектов переменной длины, состоящих из букв. Хочется преобразовать этот набор символов в фиксированное количество признаков.

#### Мешок слов

Существует набор данных под названием 20newsgroups. В нём требуется классифицировать тексты по 20 тематикам. Для примера можно рассмотреть следующий текст:

> I am thinking about getting an Infiniti G20. In consumer reports it is ranked high in many catagories including highest in reliability index for compact cars. Mitsubushi Galant was second followed by Honda Accord). A couple of things though:
> 1) In looking around I have yet to see anyone driving this car. I see lots of Honda's and Toyota's.
> 2) There is a special deal where I can get an Infinity G20, fully loaded, at dealer cost (I have check this out and the numbers match up). They are doing this because they are releasing and update mid-1993 version (includes dual air-bags) and want to get rid of their old 1993's.

В нём встречаются названия марок автомобилей, что наводит на мысль о том, что тематика текста — машины. При этом порядок слов неважен, важно наличие ключевых слов.
Ещё один пример:

> Announcing. . . Announcing. . . Announcing. . .Announcing. . .
>
> CELEBRATE LIBERTY!
> 1993 LIBERTARIAN PARTY NATIONAL CONVENTION
> AND POLITICAL EXPO
> THE MARRIOTT HOTEL AND THE SALT PALACE
> SALT LAKE CITY, UTAH
> INCLUDES INFORMATION ON DELEGATE DEALS!
> (Back by Popular Demand!)
>
> The convention will be held at the Salt Palace Convention Center and the Marriott Hotel, Salt Lake City, Utah. The business sessions, Karl Hess Institute, and Political Expo are at t

В этом тексте присутствуют названия политических партий и собраний. Можно догадаться, что текст о политике. И снова для определения тематики важна не структура документа, а то, какие слова в него входят.
Именно на этом основан подход под названием мешок слов. В нём текст кодируется в виде количества вхождений каждого слова из словаря в документ. Этот метод хорошо себя зарекомендовал, и часто используется в задачах машинного обучения. Подробнее этот он будет обсуждаться в следующих видео.

## 6.2. Предобработка текста

В этой части речь пойдёт о том, какие преобразования имеет смысл совершить над текстом, прежде чем извлекать признаки и строить модели. В частности, будут рассмотрены два важных этапа предобработки: токенизация (разбиение текста на отдельные «слова») и нормализация (приведение слов к начальной форме).

### 6.2.1. Токенизация

Как уже было упомянуто выше, токенизация — это разбиение непрерывной строки на отдельные «слова». Для демонстрации этого процесса ниже приведён текст из википедии, объясняющий, что такое текст:

> Текст (от лат. textus— «ткань; сплетение, связь, паутина, сочетание») — зафиксированная на каком-либо материальном носителе человеческая мысль; в общем плане связная и полная последовательность символов.

Видно, что этот отрывок состоит не только из букв, но и из символов: скобок, кавычек, тире. Кажется логичным удалить их, то есть избавиться от всего, что не является буквами или цифрами:

> Текст (от лат. textus–«ткань; сплетение, связь, паутина, сочетание»)— зафиксированная на каком-либо материальном носителе человеческая мысль;в общем плане связная и полная последовательность символов.

Если все их заменить на пробелы, то слова, разделённые пробелами, можно объявить отдельными токенами. Однако возникает много нюансов, например, слово «каком-либо». При замене дефиса на пробел оказывается, что есть два слова: «каком» и «либо». На самом деле это одно слово, которое было бы правильнее не разделять.

Итак, токенизация состоит из нескольких этапов. В первую очередь текст приводится к нижнему регистру. При этом, опять же, можно потерять часть информации. Например, «ООО» может являться сокращением (общество с ограниченной ответственностью), а «ооо» — просто выражением эмоций.

Следующий этап — это замена всех знаков препинания и прочих символов на пробелы. И снова на этом этапе возникает много нюансов. Как было упомянуто выше, при наличии сложных составных слов (например, «красно-чёрный») заменять в них дефис на пробел не очень разумно: может потеряться смысл слова. Кроме того, удаление знаков препинания приводит к удалению смайликов. Они могут играть большую роль при анализе твитов или записей в соц.сетях, например, для определения эмоционального окраса текста.

После этого каждое слово, отделённое пробелом, объявляется отдельным токеном. Здесь тонкость заключается в том, что некоторые наборы слов должны рассматриваться как одно. Например, названия городов («Нижний Новгород»), или сокращения («к.т.н.», кандидат технических наук, это полезный термин при рассмотрении трёх букв вместе, но по отдельности они никакой информации не несут).

Стоит отметить, что токенизация не всегда проходит так просто. В некоторых языках не принято ставить пробелы. Например, в китайском предложение — это единый набор иероглифов. Чтобы разделить его на отдельные слова, нужно применять специальные алгоритмы сегментации текста, которые находят в нём разделители по некоторым правилам.

### 6.2.2. Нормализация

Следующий этап после токенизации — это нормализация слов в тексте, то есть приведение каждого слова к его начальной форме. Например, слово «машинное» необходимо привести к слову «машинный», «шёл» — «идти». Форма слова не всегда несёт в себе полезную информацию, при этом в задачах, где данных не очень много, хотелось бы использовать как можно меньшее число признаков. Их количество напрямую зависит от количества различных слов, поэтому было бы здорово сократить их число. Приведение слов к нормальной форме позволяет это сделать.

Существует два основных подхода к нормализации: стемминг и лемматизация.

#### Стемминг

Стемминг — это самый простой метод нормализации. Его суть состоит в том, что слова «стригутся»: по некоторым правилам от каждого слова отрезается его окончание. К сожалению, данный подход не всегда работает. Например, некоторые слова при изменении формы меняются целиком: «был», «есть», «будет». Понятно, что из этих трёх слов невозможно получить одно, отрезая буквы.

#### Лемматизация

По описанной выше причине чаще пользуются более сложным подходом, лемматизацией. В нём используется словарь, в котором уже записано большое количество слов и их форм. В первую очередь слово проверяется по словарю. Если оно там есть, то понятно, к какой форме его приводить. Иначе по определённому алгоритму выводится способ изменения данного слова, на основании него делаются выводы о начальной форме.

Этот подход работает лучше, и он подходит для новых неизвестных слов. Однако лемматизация сложнее, поэтому работает гораздо медленнее, чем стемминг.

## 6.3. Извлечение признаков из текста

Далее речь пойдёт об извлечении признаков из предобработанного текста.

### 6.3.1. Счётчики слов

Как уже отмечалось ранее, для текстов очень неплохо работает подход под названием мешок слов. В этом методе текст рассматривается как неупорядоченный набор слов.

Более формально описать этот метод можно следующим образом. Пусть всего в выборке $N$ различных слов: $w_1, \ldots, w_N$. В этом случае каждый текст кодируется с помощью $N$ признаков, причём признак $j$ — это доля вхождений слова $w_j$ среди всех вхождений слов в документе.

Пусть выборка состоит из двух текстов:

> “текст состоит из слов”,
> “вхождения данного слова среди всех слов.”

Из этих текстов необходимо удалить слова «из» и «среди». Это так называемые стоп-слова, о которых позже будет рассказано подробнее.

Получается, что первый текст состоит из трёх слов, он кодируется вектором признаков, который находится в первой строке в таблице 6.1. Сумма значений всех признаков составляет единицу. Второй текст состоит из пяти слов, однако слово «слово» входит в него два раза, поэтому значение соответствующего признака составляет 0.4, для остальных признаков это значение равно 0.2 (строка 2 в таблице 6.1).

| | текст | состоит | слово | вхождение | данный | все |
|---|---|---|---|---|---|---|
| текст 1 | 0.33 | 0.33 | 0.33 | 0 | 0 | 0 |
| текст 2 | 0 | 0 | 0.4 | 0.2 | 0.2 | 0.2 |

Таблица 6.1: Частоты слов в текстах выборки

Используя такой подход, имеет смысл обращать внимание на два нюанса. Первый — это стоп-слова. Это популярные слова, встречающиеся в каждом тексте (например, предлоги или союзы) и не несущие в себе никакой информации, а только засоряющие признаки. Их стоит удалять ещё при предобработке, например, на этапе токенизации. Кроме того, имеет смысл удалять редкие слова. Если какое-то слово входит только в 1 или 2 текста, то, скорее всего, не получится его значимо учесть в модели и оценить, какой вклад оно вносит в целевую переменную.

### 6.3.2. TF-IDF

TF-IDF — это чуть более сложный подход к формированию вектора признаков. Как и в предыдущем методе, считается, что если слово часто встречается в тексте, и оно не является стоп-словом, то, скорее всего, оно важно. Но есть и вторая тонкость. Если слово встречается в других документах реже, чем в данном, то и в этом случае, скорее всего, оно важно для текста. По этому слову можно отличить этот текст от остальных. Если учесть описанные выше соображения, в результате получится подход TF-IDF.

Значение признака для слова $w$ и текста $x$ вычисляется по следующей формуле:

$$\text{TF-IDF}(x, w) = n_{dw} \log \frac{l}{n_w}.$$

Эта формула состоит из двух частей. $n_{dw}$ — доля вхождений слова $w$ в документ $d$. Если слово часто встречается в тексте, то оно важно для него, и значение признака будет выше. Второй множитель называется IDF (inverse document frequency, обратная документная частота), это отношение $l$, общего количества документов, к $n_w$, числу документов в выборке, в которых слово $w$ встречается хотя бы раз. Если это отношение велико, слово редко встречается в других документах, значение признака будет увеличиваться. Если слово встречается в каждом тексте, то значение признака будет нулевым ($\log \frac{l}{l} = 0$).

## 6.4. Извлечение признаков из текста - 2

### 6.4.1. Учёт порядка токенов

Метод «мешок слов», о котором шла речь ранее, никак не учитывает порядок слов в документе, а лишь подсчитывает, сколько раз каждое слово встретилось в тексте.

Этот подход не очень хорош. Пусть в тексте есть слова «нравится» и «не нравится», у них противоположный смысл. «Мешок слов» никак не поможет уловить эти смыслы: будет только информация о том, что в тексте есть слово «нравится» и частица «не». Это не несёт много информации.

Более того, если в тексте учитывать не только слова, но и словосочетания, то признаковое пространство становится более обширным. При этом появляется возможность находить более сложные закономерности, используя простые модели (аналогично при добавлении новых признаков линейные модели могут находить более сложные разделяющие поверхности).

### 6.4.2. N-граммы

Использование n-грамм — этосамый простой способ учитывать порядок слов. По сути n-грамма — это последовательность из $n$ идущих подряд слов в тексте. Для предложения «Наборы подряд идущих токенов» существуют следующие n-граммы:

**униграммы**$(n = 1)$**:** наборы, подряд, идущих, токенов;

**биграммы** $(n = 2)$ : наборы подряд, подряд идущих, идущих токенов;

**триграммы** $(n = 3)$ : наборы подряд идущих, подряд идущих токенов.

После того как в тексте найдены все требующиеся n-граммы, используются те же подходы, что и для мешка слов: подсчитываются счётчики, вычисляются TF-IDF, либо что-то ещё.

Чем больше $n$, до которого будут найдены n-граммы, тем больше признаков получится, призна ковое пространство будет более богатым. При этом $n$ — это гиперпараметр, и его увеличение может привести к переобучению. Например, если $n$ — это длина текста, то у каждого документа будет уникальный признак, и алгоритм сможет переучиться под обучающую выборку.

### 6.4.3. Буквенные n-граммы

N-граммы можно использовать не только на словах. В качестве токенов можно рассматривать отдельные символы в предложении. После нахождения таких токенов для них можно также вычислять n-граммы (буквенные n-граммы). В качестве признаков, как и раньше, могут выступать счётчики или TF-IDF.

У этого подхода есть много преимуществ. Он позволяет учитывать смайлы или известные слова в незнакомых формах. Часто буквенные n-граммы используют вместе с n-граммами по словам.

### 6.4.4. Skip-граммы

Skip-граммы — это чуть более расширенный подход к использованию n-грамм. Более точно они называются $k$-skip-$n$-граммы, это наборы из $n$ токенов, причём расстояние между соседними должно составлять не более $k$ токенов. В качестве примера можно снова рассмотреть предложение «Наборы подряд идущих токенов». Для него:

**Биграммы:** наборы подряд, подряд идущих, идущих токенов;

**1-skip-2-граммы:** наборы подряд, подряд идущих, идущих токенов, **наборы идущих**, **подряд токенов**. В этом случае расстояние между токенами не должно превышать 1 слово, поэтому к предыдущему набору добавляются следующие пары слов: наборы идущих (между токенами 1 слово), подряд токенов (между токенами 1 слово);

### 6.4.5. Хэширование

Ещё один подход, часто использующийся для вычисления признаков на текстах, — это хэширование.

Пусть функция $h(x)$ принимает на вход и выдаёт некоторый хэш, причём возможное количество выходов функции составляет $2^n$, $n$— некоторое небольшое число. Все слова $x$ в тексте можно заменить на их хэши $h(x)$, а далее использовать их в качестве токенов и вычислять для них счётчики, TF-IDF или что-то другое.

Заменить слова на их хэши просто, тем не менее, у этого подхода есть свои преимущества. Он позволяет сократить количество признаков: можно не задумываться о том, какие слова объединить в один признак, а объединять слова, имеющие одинаковые хэши. Понятно, что это не очень умный подход, ведь слова можно было пытаться группировать по смыслу, но он неплохо работает и не требует никаких усилий. Но самое главное преимущество при использовании хэширования — отсутствие необходимости запоминать соответствие

между словами и номерами признаков. Не нужно помнить, например, что слово «токен» — это десятый признак в используемом признаковом пространстве, нужно лишь вычислить хэш, его значение и будет индексом признака слова. Это существенно упрощает хранение модели.

## 6.5. Обучение моделей на текстах

### 6.5.1. Подготовка выборки

Как уже было сказано ранее, прежде чем начать обучать модель, необходимо подготовить текст: сформировать выборку и вычислить признаки. Как правило, для этого сначала из текста удаляются слишком редкие или популярные слова (стоп-слова). Затем вычисляются признаки. Это могут быть $n$-граммы, skip-граммы, на которых вычисляются счётчики или TF-IDF.

Как правило, размерность полученного пространства признаков оказывается очень большой. При использовании униграмм количество признаков может составлять $10^3$ – $10^4$, если добавить биграммы и триграммы, то при использовании большого корпуса текстов количество признаков может достигать $10^6$ – $10^7$. Поэтому можно пробовать отбирать признаки, находить только те униграммы, биграммы и т.д., которые коррелируют с целевой переменной. Кроме того, можно понижать размерность. Например, использовать метод главных компонент для перехода в пространство меньшей размерности, в котором всё ещё будет сохраняться хорошее признаковое описание.

### 6.5.2. Обучение модели

Скорее всего, размерность признакового пространства не получится радикально уменьшить, и признаков будет очень много. Нужно понять, как в такой ситуации будут работать различные методы машинного обучения.

В случайном лесе используются деревья большой глубины, они строятся до тех пор, пока в каждом листе не окажется очень мало объектов. Если признаков очень много, то этот подход работает не очень хорошо: деревья будут очень глубокими, на их построение будет уходить слишком много времени, и можно просто не дождаться, когда лес достроится до нужного размера.

При использовании градиентного бустинга решающие деревья, наоборот, имеют очень небольшую глубину. Но это тоже проблема. Каждое дерево может учесть лишь небольшое подмножество признаков, в то время как зачастую ответ зависит от комбинации большого количества слов в документе. Поэтому для хорошей работы градиентного бустинга нужно будет использовать очень много деревьев, но даже в этом случае нет гарантии, что полученное качество будет приемлемым.

Байесовские методы обычно хорошо себя показывают при работе на текстах. В частности, работа наивного байесовского классификатора была продемонстрирована ранее в задаче детекции спама.

Чаще всего на текстовых данных используются линейные модели. Они хорошо масштабируются, могут работать с большим количеством признаков, на очень больших выборках.

### 6.5.3. Линейные модели и SGD

Для оптимизации линейных моделей может использоваться стохастический градиентный спуск. Он позволяет считывать объекты с диска по одному, и для каждого из них делать градиентный шаг. Более того, этот подход становится совсем прост в реализации, если используется хэширование:

```
пока не выполнен критерий останова:
    t = следующий текст
    для всех слов x в t:
        w_{h(x)} = w_{h(x)} − α∇_{w_{h(x)}} Q(w).
```