Токенезация

НЛП включает в себя большое разнообразие процедур. Токенизация является одним из них. Основная задача состоит в том, чтобы разбить последовательность символов на единицы, называемые токенами. Токены обычно представлены словами, числами или знаками препинания. Иногда они могут быть представлены предложениями или морфемами (частями слова). Токенизация — это первый шаг в предварительной обработке текста. Это очень важная процедура; прежде чем переходить к более сложным процедурам НЛП, нам нужно определить слова, которые могут помочь нам интерпретировать значение.

Проблемы с токенизацией

Основная проблема заключается в выборе правильного токена. Это просто, но сложно. Давайте проанализируем пример ниже.

```
"There are a lot of cats in my house."

["There", "are", "a", "lot", "of", "cats", "in", "my", "house"]
```

Этот пример тривиален; мы используем пробелы, чтобы разделить предложение на токены и избавиться от этой точки в конце. Но иногда английский язык отображает менее очевидные падежи. Что нам делать с апострофами или комбинацией цифр и букв?

```
    "we're"
1.  "we're"
2.  ["we", "re"]
3.  "were"
4.  ["we'", "re"]
5.  ["we", "'re"]
6.  ["we", "'", "re"]
```

Какой токен здесь наиболее подходящий? Интуитивно мы можем сказать, что нам следует выбрать первый вариант. Конечно, второй вариант также имеет смысл, поскольку «we're» — это сокращение от «we are».

Все остальные варианты также теоретически возможны.

Что, если мы говорим о городе со сложным названием, например, `New York` или `["New", "York"]?` ?

Как видите, выбор правильного токена может оказаться непростой задачей. Правильного ответа нет, поэтому имеет смысл выбрать наиболее подходящий токенизатор и продолжить работу с ним.

Токенизация в NLTK

NLTK имеет модуль токенизации, который состоит из разных подмодулей. Мы рассмотрим наиболее значимые из них. В приведенной ниже таблице описаны некоторые из них. Первый столбец содержит имена токенизаторов. Чтобы импортировать конкретный, используйте from `nltk.tokenize import <tokenizer>`. Вот несколько примеров импорта:

```
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
```

Модуль	Описание
word_tokenize()	Возвращает слова и знаки препинания.
WordPunctTokenizer()	Возвращает токены из строки буквенных или неалфавитных символов.
regexp_tokenize()	Возвращает токены, используя стандартные регулярные выражения.
TreebankWordTokenizer()	Возвращает токены, используя регулярные выражения.
sent_tokenize()	Возвращает токенезированные предложения.

Токенизация слов

Давайте посмотрим на пример. Представьте, что у нас есть строка из трех предложений:

```
text = "I have got a cat. My cat's name is C-3PO. He's golden."
```

Теперь давайте посмотрим на каждый метод токенизации из таблицы. Не забудьте импортировать их все заранее.

В приведенном ниже примере мы передаем текстовую переменную `text` методу `word_tokenize()`:

```
print(word_tokenize(text))
# ['I', 'have', 'got', 'a', 'cat', '.', 'My', 'cat', "'s", 'name', 'is',
'C-3PO', '.', 'He', "'s", 'golden', '.']
```

Результатом является список строк (токенов). Функция разбивает строку на слова и знаки препинания. Имейте в виду притяжательные и сокращения. Токенизатор преобразует все `'s` в отдельные слова.

Конечно, мы понимаем, что `cat's` тоже могут быть признаны одним токеном.

Следующий фрагмент кода с `WordPunctTokenizer()`. Этот токенизатор похож на первый, но результат немного отличается. Все знаки препинания, включая тире и апострофы, являются отдельными лексемами. Теперь `C-3PO`, имя кота, разделено на три токена. В данном случае такое поведение не является оптимальным.

```
wpt = WordPunctTokenizer()
print(wpt.tokenize(text))
```

```
# ['I', 'have', 'got', 'a', 'cat', '.', 'My', 'cat', "'", 's', 'name',
'is', 'C', '-', '3PO', '.', 'He', "'", 's', 'golden', '.']
```

В следующем примере показаны результаты `TreebankWordTokenizer()`.

```
tbw = TreebankWordTokenizer()
print(tbw.tokenize(text))
# ['I', 'have', 'got', 'a', 'cat.', 'My', 'cat', "'s", 'name', 'is', 'C-
3PO.', 'He', "'s", 'golden', '.']
```

`TreebankWordTokenizer()` работает почти так же, как `word_tokenize()`. Обратите внимание на точки — они образуют лексему с предыдущим словом, но последняя точка — это отдельная лексема.

`Word_tokenize()`, напротив, во всех случаях распознает точки как отдельные токены. Более того, апостроф и `s` не разделены, как в `WordPunctTokenizer()`.

Теперь перейдем к следующему способу. Функция `regexp_tokenize()` использует регулярные выражения и принимает два аргумента: строку и шаблон для токенов.

```
# 1
print(regexp_tokenize(text, "[A-z]+"))
# ['I', 'have', 'got', 'a', 'cat', 'My', 'cat', 's', 'name', 'is', 'C',
'PO', 'He', 's', 'golden']

# 2
print(regexp_tokenize(text, "[0-9A-z]+"))
# ['I', 'have', 'got', 'a', 'cat', 'My', 'cat', 's', 'name', 'is', 'C',
'3PO', 'He', 's', 'golden']

# 3
print(regexp_tokenize(text, "[0-9A-z']+"))
# ['I', 'have', 'got', 'a', 'cat', 'My', "cat's", 'name', 'is', 'C',
'3PO', "He's", 'golden']

# 4
print(regexp_tokenize(text, "[0-9A-z'\-]+"))
# ['I', 'have', 'got', 'a', 'cat', 'My', "cat's", 'name', 'is', 'C-3PO',
"He's", 'golden']
```

Шаблон `[A-z]+` в первом примере выше позволяет нам найти все слова или буквы, но оставляет в стороне целые числа и знаки препинания. Из-за этого все притяжательные формы и кошачье имя разделены.

Следующий шаблон улучшает поиск токенов по мере добавления целых чисел. Это улучшает поиск имени кота, но способ не оптимален.

Третий шаблон с апострофом также позволяет токенизатору находить притяжательные формы.

В последнем шаблоне есть дефис, поэтому имя кота распознается без ошибок.

Вы видите, что получение токенов с помощью регулярных выражений может быть гибким. Мы меняем шаблон в каждом случае; это позволяет получить более точные результаты.

Токенизация предложения

Наконец, давайте посмотрим на модуль `sent_tokenize()`. Он разбивает строку на предложения:

```
print(sent_tokenize(text))
# ['I have got a cat.', "My cat's name is C-3PO.", "He's golden."]
```

Токенизация предложений также является сложной задачей. Точка, например, может обозначать аббревиатуры или сокращения, а не только конец предложения. Давайте посмотрим на примеры.

```
text_2 = "Mrs. Beam lives in the U.S.A., it is her motherland. She lost about 9 kilos (20 lbs.) last year."
print(sent_tokenize(text_2))
# ['Mrs. Beam lives in the U.S.A., it is her motherland.', 'She lost about 9 kilos (20 lbs.)', 'last year.']
```

`Send_tokenize()` включает в себя список типичных сокращений и сокращений с точками, поэтому они не распознаются как конец предложения. Иногда это все еще дает запутанные результаты.

Например, после токенизации `text_2`, `.)` был распознан как конец предложения. Это ошибка. Последняя часть в выводе токенизатора — `«last year.»`. но оно должно принадлежать предыдущему предложению.

Если вы имеете дело с неформальными текстами, такими как комментарии, разделение их на предложения может быть особенно проблематичным. Например, в `text_3` много точек и нет пробелов, поэтому два предложения распознаются как одно.

```
text_3 = "The plot of the film is cool!!!!!!! but the characters leave much to be desired....i don't like them."
print(sent_tokenize(text_3))
# ['The plot of the film is cool!!!!!!!', "but the characters leave much to be desired....i don't like them."]
```

Подводя итог, можно сказать, что токенизация — важная процедура предварительной обработки текста в НЛП. В этой теме вы узнали: • Основные условия и сложности токенизации; • Как разбить текст на слова с помощью разных модулей NLTK; • Как разбить текст на предложения с помощью модуля `sent_tokenize()`.