Theory: Text corpora in NLTK

28 minutes 0 / 5 problems solved

Many NLP tasks involve operations with various corpora. But what is a **corpus?** This term stands for a large and structured collection of linguistic data that represent specific features of one or multiple languages. Corpora are widely applied in statistical linguistic analysis, and in some way, it is an equivalent of datasets; we use them to train algorithms and models.

In this topic, we will discuss which text corpora are included in NLTK. You can access them using the import construction `from nltk.corpus import <corpus>`.

§1. Plaintext vs annotated corpora

In general, all text collections (or **corpora**) can be divided into **plaintext** and **annotated** ones. You can always tell the difference. An **annotated corpus** contains additional labels, while a **plaintext** one has none of them.

Let's start with plaintext corpora. For the most part, files that make up such collections have no XML or other tags inside. A good example of such a plaintext corpus would be the Project Gutenberg dataset. Do not forget to download the corpora before you start working with them!

```
import nltk

nltk.download('gutenberg')
```

With the help of the `raw()` method, we can see the unprocessed texts. In our case, it will be *'Alice's Adventures in Wonderland',* the novel integrated with this collection.

```
from nltk.corpus import gutenberg

print(gutenberg.raw('carroll-alice.txt'))
# [Alice's Adventures in Wonderland by Lewis Carroll 1865]
#
# CHAPTER I. Down the Rabbit-Hole
#

# Alice was beginning to get very tired of sitting by her sister on the bank, and
of having nothing to do...
```

We can also apply `sents()` to obtain a list of sentences and further index or slice the list in order to get particular ones:

```
print(gutenberg.sents('carroll-alice.txt')[1:3])
# [['CHAPTER', 'I', '.'], ['Down', 'the', 'Rabbit', '-', 'Hole']]
```

To achieve more accurate results in our NLP tasks, however, we may need to draw up a collection in advance. It often implies assigning certain labels to every word, sentence, or other units of a text. This procedure is called **annotation** and it is widely used as an elementary step in machine learning. It simplifies retrieving patterns and features.

One of the basic examples of the annotation is part-of-speech tagging. Let's look at how it is done in the Brown corpus:

```
from nltk.corpus import brown

print(brown.tagged_words()[:5])
# [('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ('Grand', 'JJ-
TL'), ('Jury', 'NN-TL')]
```

`tagged_words()` is used to retrieve information about the words in the corpus. As you can see, the result is a list of tuples — each contains a word and its grammatical tag.

Current topic:

Text corpora in NLTK

Topic depends on:

Stage 1
Taking input 17

Slicing

Topic is required for:

Word2Vec

Table of contents:

↑ Text corpora in NLTK

§1. Plaintext vs annotated corpora

§2. Types of annotation

§3. Corpus methods in NLTK

§4. Creating your own basic corpus

§5. Conclusion

Feedback & Comments

Of course, there are more methods for interacting with corpus data. We will discuss them (as well as annotation) further in the next sections.

§2. Types of annotation

Now, let's go over the main types of annotated corpora in NLTK and see where they can be applied.

POS tagged corpora can be used for a number of tasks; other types of annotation usually depend on this one. NLTK includes such tagged corpora for English as well as for other languages. You have seen an example of such a corpus above.

Chunked corpora can help in named-entity recognition (NER). This procedure includes extraction of terms referring to real-world objects: locations, organizations, people, and others. For example, we might detect the phrase *Trafalgar Square* as a named entity and then classify it as a location.

The CoNLL 2000 corpus is responsible for phrasal chunks in NLTK. The word clusters are formed not only if they relate to the real-world objects but also if they are bound by syntax, for example, a noun or a verb phrase (*NP* or *VP*). Such units are called **chunks**, this is why these types of corpora are called chunked. Let's look at the example from CoNLL 2000 using `chunked_sents()`:

```
from nltk.corpus import conll2000

print(conll2000.chunked_sents()[1])
# ...
# (NP a/DT firm/NN monetary/JJ policy/NN)
# (VP has/VBZ helped/VBN to/TO prevent/VB)
# (NP a/DT freefall/NN)
# ...
```

Here we have three phrasal chunks: *a firm monetary policy*, *has helped to prevent*, and *a freefall*. We can also see a part of speech indicated after each word.

Now, let's turn our gaze to the types of corpora that can be used for **word sense disambiguation** (WSD). The idea behind WSD is to state the most suitable meaning for a word in case of its ambiguity in a sentence. Consider the following example: *The second room was used as a study*. We understand that here the word *study* means a room designed for academic work, such as writing. However, for NLP systems, it may not be clear: the most frequent meaning refers to an act of gaining knowledge.

Imagine, we need to differentiate *"study"* as a noun from *"study"* as a verb. Bear in mind that syntactic features of a word can also be changed with its grammatical category. We may have to use **syntactically parsed corpora** first. NLTK has several of them, the Penn Treebank, for example, it is one of the most popular. Let's take a look at its annotation via `parsed_sents()`:

```
from nltk.corpus import treebank

print(treebank.parsed_sents()[35])
# (S
#   (NP-SBJ (NN Compound) (NNS yields))
#   (VP
#     (VBP assume)
#     (UCP
#       (NP (NP (NN reinvestment)) (PP (IN of) (NP (NNS dividends))))

#       (CC and)

# ...
```

Each word is annotated according to the Penn Treebank POS tagset. A tag stands right before the word, in the same parentheses. For example, `(NN reinvestment)` means that *reinvestment* is a singular common noun. We can see *NP* and *VP,* with which we have dealt in CoNLL 2000, and *PP* that refers to a *prepositional phrase*. These labels tell us about the syntax of the sentence. We can also observe *S*, indicating that it is a declarative clause,

and *-SBJ*, specifying the place of the grammatical subject. If you want to understand the annotation better, take a look at the guidelines for this corpus.

As the second step in WSD, we need to use a **sense tagged corpus**. Such a corpus is annotated according to the sense of a word in a context. NLTK has SemCor, which is the subcorpus of the Brown collection. It includes both part-of-speech and sense labels. With the help of `tagged_sents()`, we will see how this kind of markup looks like.

```
from nltk.corpus import semcor

print(semcor.tagged_sents(tag='sem'))

# [[['The'], Tree(Lemma('group.n.01.group'), [Tree('NE', ['Fulton', 'County', 'Gra
nd', 'Jury'])]), ...]
```

We specified `'sem'` so that we could get a semantic tag instead of a default POS one. In our case, the tag is `'group.n.01.group'`. It is placed right before the named-entity construction it describes.

The sense inventory of SemCor is based on WordNet, the English linguistic database. In NLTK, WordNet itself is presented as an advanced corpus, which contains information about synonyms and antonyms of words, as well as their definitions, and more.

We can also find **topic**, **genre**, or **polarity annotated corpora** in the library. They are used for text classification and sentiment analysis. Among other categorized corpora, NLTK has the classic Reuters-21578 collection of news documents.

There are also standalone datasets with their own unique annotations, for instance, the CMU Pronunciation Dictionary with phonetic transcriptions provided. You can learn more about the corpora available in NLTK Corpus Reader Manual (Object section).

§3. Corpus methods in NLTK

In this section, we will cover the main methods for working with corpus data.

Most of the collections, including the ones we have used previously, are composed of a set of files. To see their contents, use the `fileids()` method:

```
from nltk.corpus import gutenberg

print(gutenberg.fileids())
# ['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-
kjv.txt', 'blake-poems.txt',
#  'bryant-stories.txt', 'burgess-busterbrown.txt', 'carroll-
alice.txt', 'chesterton-ball.txt',
#  'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-
parents.txt', 'melville-moby_dick.txt',
#  'milton-paradise.txt', 'shakespeare-caesar.txt', 'shakespeare-
hamlet.txt', 'shakespeare-macbeth.txt',
#  'whitman-leaves.txt']
```

Depending on the type of corpus, there can be different ways of reading the data. The standard interface examples are shown in the table below:

Method	Application
`raw()`	Accesses a raw text from a plain corpus.
`words()`	Divides a corpus into words.
`sents()`	Divides a corpus into sentences.
`paras()`	Divides a corpus into paragraphs.
`readme()`	Accesses a README file for a corpus, if found.

In the brackets of the first four methods, we can specify the files that we are looking into. This is exactly what we have done with the `raw()` method when we retrieved information for the '*carroll-alice.txt*' file from the Gutenberg corpus.

In addition to the standard methods, there are annotation-based methods. The most common of them are listed below:

Method	Application
`tagged_words()`	Displays each word in a text with its POS tag.
`tagged_sents()`	Displays each word in a sentence with its POS tag.
`tagged_paras()`	Displays each word in a paragraph with its POS tag.
`chunks()`	Accesses the whole corpus broken into chunks.
`chunked_sents()`	Displays chunk structures for a sentence.
`parsed_sents()`	Shows the syntactic structure for a sentence.
`parsed_paras()`	Shows syntactic structure for sentences in a paragraph.
`categories()`	Shows categories, such as genres or polarity values.

Let's illustrate a couple of them with examples. In the previous section, we mentioned `chunked_sents()` while working with one of the parsed corpora. Below is an example of the `chunks()` method. *Southwest Conference* is a single chunk here, so the words are contained in a separate list.

```
from nltk.corpus import semcor

print(semcor.chunks('brown1/tagfiles/br-a12.xml')[17:19])
# [['in'], ['Southwest', 'Conference']]
```

As for a categorized corpus, it goes like this:

```
from nltk.corpus import movie_reviews

print(movie_reviews.categories())
# ['neg', 'pos']

print(movie_reviews.words(categories='pos')[:10])

# ['films', 'adapted', 'from', 'comic', 'books', 'have', 'had', 'plenty', 'of', 's
uccess']
```

As you can see, with these methods, we can also indicate specific files or subcorpora.

§4. Creating your own basic corpus

Apart from using corpora available in NLTK, you may want to create your own. To load your raw text collection as a corpus, use `PlaintextCorpusReader`. In the example below, we assume that the whole collection is stored in the `/home/user/corpora` folder. Then we pass it as the first argument to the Corpus Reader, the second argument of which is a pattern that matches all *txt* files in our folder:

```
from nltk.corpus import PlaintextCorpusReader

root = '/home/user/corpora/'
corpus = PlaintextCorpusReader(root, '.*\.txt')
```

Once we have done this, we can check the files with `fileids()` and use other methods that can be applied to plaintext corpora.

Raw text collections are not the only type of collections we can create in NLTK. Different corpus formats require different reader classes, for example, `TaggedCorpusReader` or `CategorizedTaggedCorpusReader`. You can find more in the NLTK Corpus Reader Manual (Classes section).

§5. Conclusion

So far, in this topic we've learned the following points:

- A corpus is a large collection of language data;
- We mostly deal with POS tagged, parsed, sense annotated or category tagged corpora in NLTK;

- There are standard and annotation-based methods for accessing the data;
- You can create your own corpus using NLTK tools; we have shown how to do it with a plaintext corpus.

Report a typo

11 users liked this theory. 0 didn't like it. What about you?

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion