Text Preprocessing With NLTK

Common Preprocessing Methods for NLP

Raymond Cheng Jun 29 · 2 min read ★

Photo by Carlos Muza on Unsplash

Intro

Almost every **Natural Language Processing (NLP)** task requires text to be preprocessed before training a model. Deep learning models cannot use raw text directly, so it is up to us researchers to clean the text ourselves. Depending on the nature of the task, the preprocessing methods can be different. This tutorial will teach the most common preprocessing approach that can fit in with various NLP tasks using **NLTK (Natural Language Toolkit)**.

Why NLTK?

- **Popularity**: NLTK is one of the leading platforms for dealing with language data.
- **Simplicity**: Provides easy-to-use APIs for a wide variety of text preprocessing methods
- **Community**: It has a large and active community that supports the library and improves it
- **Open Source**: Free and open-source available for Windows, Mac OSX, and Linux.

Now you know the benefits of NLTK, let's get started!

Tutorial Overview

1. Lowercase
2. Removing Punctuation
3. Tokenization
4. Stopword Filtering
5. Stemming
6. Part-of-Speech Tagger

All code displayed in this tutorial can be accessed in my Github repo.

Import NLTK

Before preprocessing, we need to first download the NLTK library.

```
pip install nltk
```

Then, we can import the library in our Python notebook and download its contents.

Import Library

```
In [1]: import nltk
        nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

```
Out[1]: True
```

nltk.ipynb hosted with ♡ by GitHub view raw

Lowercase

As an example, we grab the first sentence from the book *Pride and Prejudice* as the text. We convert the sentence to lowercase via `text.lower()` .

Lowercase

```
In [2]: text = "It is a truth universally acknowledged, that a single man
         in possession of a good fortune, must be in want of a wife."
        text = text.lower()
        print(text)
```

```
it is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife.
```

lowercase.ipynb hosted with ♡ by GitHub view raw

Removing Punctuation

To remove punctuation, we save only the characters that are not punctuation, which can be checked by using `string.punctuation` .

Removing Punctuation

In [3]:
```
import string
print(string.punctuation)
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

In [4]:
```
text_p = "".join([char for char in text if char not in string.punctuation])
print(text_p)
```

```
it is a truth universally acknowledged that a single man in possession of a good fortune must be in want of a wife
```

punctuation.ipynb hosted with ♡ by GitHub view raw

Tokenization

Strings can be tokenized into tokens via `nltk.word_tokenize` .

Word Tokenization

```
In [5]: from nltk import word_tokenize
        words = word_tokenize(text_p)
        print(words)
```

```
['it', 'is', 'a', 'truth', 'universally', 'acknowledged', 'that',
'a', 'single', 'man', 'in', 'possession', 'of', 'a', 'good', 'fort
une', 'must', 'be', 'in', 'want', 'of', 'a', 'wife']
```

tokenization.ipynb hosted with ♡ by GitHub view raw

Stopword Filtering

We can use `nltk.corpus.stopwords.words('english')` to fetch a list of stopwords in the English dictionary. Then, we remove the tokens that are stopwords.

Stopword Filtering

```
In [6]: from nltk.corpus import stopwords
        stop_words = stopwords.words('english')
        print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'yo
u', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yours
elf', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's",
'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they',
'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who',
'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is',
'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and',
'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at',
'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throu
gh', 'during', 'before', 'after', 'above', 'below', 'to', 'from',
'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again',
'further', 'then', 'once', 'here', 'there', 'when', 'where', 'wh
```

```
y', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'o
ther', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same',
'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'do
n', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 'r
e', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'did
n', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "has
```

stopword.ipynb hosted with ♡ by GitHub view raw

Stemming

We stem the tokens using `nltk.stem.porter.PorterStemmer` to get the stemmed tokens.

Stemming

In [8]:
```
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()
stemmed = [porter.stem(word) for word in filtered_words]
print(stemmed)
```

```
['truth', 'univers', 'acknowledg', 'singl', 'man', 'possess', 'goo
d', 'fortun', 'must', 'want', 'wife']
```

stemming.ipynb hosted with ♡ by GitHub view raw

POS Tagger

Lastly, we can use `nltk.pos_tag` to retrieve the part of speech of each token in a list.

POS

In [9]:
```
from nltk import pos_tag
pos = pos_tag(filtered_words)
print(pos)
```

```
[('truth', 'NN'), ('universally', 'RB'), ('acknowledged', 'VBD'),
('single', 'JJ'), ('man', 'NN'), ('possession', 'NN'), ('good', 'J
J'), ('fortune', 'NN'), ('must', 'MD'), ('want', 'VB'), ('wife',
'NN')]
```

pos.ipynb hosted with ♡ by GitHub view raw

The full notebook can be seen here.

Combining all Together

We can combine all the preprocessing methods above and create a `preprocess` function that takes in a *.txt* file and handles all the preprocessing. We print out the tokens, filtered words (after stopword filtering), stemmed words, and POS, one of which is usually passed on to the model or for further processing. We use the *Pride and Prejudice* book (accessible here) and preprocess it.

This notebook can be accessed here.

Conclusion

Text preprocessing is an important first step for any NLP application. In this tutorial, we discussed several popular preprocessing approaches using NLTK: lowercase, removing punctuation, tokenization, stopword filtering, stemming, and part-of-speech tagger.