Part 3: Step by Step Guide to NLP – Text Cleaning and Preprocessing

SAVE 25,500 ($400) on Certified AI & ML BlackBelt Plus Program Limited period Offer!! Enroll NOW 01 D 16 H 45 M 35 S

Chirag Goyal — June 15, 2021

Advanced NLP Project Python Text

This article was published as a part of the Data Science Blogathon

Introduction

This article is part of an ongoing blog series on Natural Language Processing (NLP). In part-1and part-2 of this blog series, we complete the theoretical concepts related to NLP. Now, in continuation of that part, in this article, we will cover some of the new concepts.

In this article, we will understand the terminologies required and then we start our journey towards text cleaning and preprocessing, which is a very crucial component while we are working with NLP tasks.

This is part-3 of the blog series on the Step by Step Guide to Natural Language Processing.

Table of Contents

1. Familiar with Terminologies

- Corpus
- Tokens
- Tokenization
- Text object
- Morpheme
- Lexicon

2. What is Tokenization?

- White-space Tokenization
- Regular Expression Tokenization
- Sentence and Word Tokenization

3. Noise Entities Removal

- Removal of Punctuation marks
- Removal of stopwords, etc.

4. Data Visualization for Text Data

- Word Cloud

5. Parts of Speech (POS) Tagging

Familiar with Terminologies

Before moving further in this blog series, I would like to discuss the terminologies that are used in the series so that you have no confusion related to terminologies:

Part 3: Step by Step Guide to NLP – Text Cleaning and Preprocessing

A Corpus is defined as a collection of text documents.

For Example,

```
A data set containing news is a corpus or
The tweets containing Twitter data are a corpus.
```

So corpus consists of documents, documents comprise paragraphs, paragraphs comprise sentences and sentences comprise further smaller units which are called Tokens.

Corpus > Documents > Paragraphs > Sentences > Tokens

Tokens

Tokens can be words, phrases, or Engrams, and Engrams are defined as the group of n words together.

For example, consider the sentence given below:

```
Sentence: I like my iphone
```

For the above sentence, the different engrams are as follows:

```
Uni-grams(n=1) are: I, like, my, iphone
Di-grams(n=2) are: I like, like my, my iphone
Tri-grams(n=3) are: I like my, like my iphone
```

So, uni-grams are representing one word, di-grams are representing two words together and tri-grams are representing three words together.

Tokenization

It is the process of converting a text into tokens.

Text object

The text object is a sentence or a phrase or a word or an article.

Morpheme

In the field of NLP, a Morpheme is defined as the base form of a word. A token is generally made up of two components,

- **Morphemes:** The base form of the word, and
- **Inflectional forms:** The suffixes and prefixes added to morphemes.

Let's discuss the structure of the tokens:

Structure of token : <prefix> <morpheme> <suffix>

For Example,

```
Consider the word: Antinationalist
```

Example : Antinationalist : Anti + national + ist

which is made up of the following components:

- Inflectional forms- Anti and ist

Part 3: Step by Step Guide to NLP – Text Cleaning and Preprocessing

Lexicon

It represents all the words and phrases used in a particular language or subject.

What is Tokenization?

Tokenization is a process of splitting a text object into smaller units which are also called tokens. Examples of tokens include words, numbers, engrams, or even symbols. The most commonly used tokenization process is White-space Tokenization.

Let's discuss the two different types of Tokenization:

- White-space Tokenization
- Regular Expression Tokenization

White-space Tokenization

It is also known as unigram tokenization. In this process, we split the entire text into words by splitting them from white spaces.

For Example, Consider the following sentence-

```
Sentence: I went to New-York to play football
```

```
Tokens generated are:  "I", "went", "to", "New-York", "to", "play", "football"
```

Notice that "New-York" is not split further because the tokenization process was based on whitespaces only.

Regular Expression Tokenization

It is another type of Tokenization process, in which a regular expression pattern is used to get the tokens.

For Example, consider the following string containing multiple delimiters such as comma, semi-colon, and white space.

```
Sentence:=  "Basketball, Hockey; Golf Tennis"
re.split(r'[;,s]', Sentence
```

```
Tokens generated are:  "Basketball", "Hockey", "Golf", "Tennis"
```

Therefore, using Regular expression, we can split the text by passing a splitting pattern.

NOTE

Tokenization can be performed at the sentence level or at the word level or even at the character level. Based on it we discuss the following two types of Tokenization:

- Sentence Tokenization
- Word Tokenization

Sentence Tokenization

Sentence tokenization, also known as **Sentence Segmentation** is the technique of dividing a string of written language into its component sentences. The idea here looks very simple. In English and some other languages, we can split apart the sentences whenever we see a punctuation mark.

However, even in English, this problem is not trivial due to the use of full stop characters for abbreviations. When processing plain text, tables of abbreviations that contain periods can help us to prevent incorrect assignment of sentence boundaries. In many cases, we use libraries to do that job for us, so don't worry too much about the details for now.

In simple words, the Sentence tokenizer breaks text paragraphs into sentences.

```
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

```
['Hello friends!.', 'Good Morning!', 'Today we will learn Natural Language Processing.', 'It is very interesting']
```

Word Tokenization

Word tokenization, also known as **Word Segmentation** is the problem of dividing a string of written language into its component words. White space is a good approximation of a word divider in English and many other languages with the help of some form of Latin alphabet.

However, we still can have problems if we only split by space to achieve the wanted results. Some English compound nouns are variably written and sometimes they contain a space. In most cases, we use a library to achieve the wanted results, so again don't worry too much about the details.

In simple words, Word tokenizer breaks text paragraphs into words.

```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Hello', 'friends', '!', '.', 'Good', 'Morning', '!', 'Today', 'we', 'will', 'learn', 'Natural', 'Language', 'Processing',
'.', 'It', 'is', 'very', 'interesting']
```

Noise Entities Removal

Noise is considered as that piece of text which is not relevant to the context of the data and the final output.

For Example,

- Language stopwords (commonly used words of a language – is, am, the, of, in, etc),
- URLs or links,
- Social media entities (mentions, hashtags),
- Punctuations, and Industry-Specific words.

The general steps which we have to follow to deal with noise removal are as follows:

- Firstly, prepare a dictionary of noisy entities,
- Then, iterate the text object by tokens (or by words),
- Finally, eliminating those tokens which are present in the noise dictionary.

Removal of Stopwords

Image Source: Google Images

Let's first understand what exactly are "Stopwords"?

learning to textual data, these words can add a lot of noise. That's why we remove these irrelevant words from our analysis. Stopwords are considered as the noise in the text.

Stop words usually refer to the most common words such as "and", "the", "a" in a language.

Note that there is no single universal list of stopwords. The list of the stop words can change depending on your problem statement.

For stopwords, we have an NLTK toolkit that has a predefined list of stopwords that refers to the most common words. If you use it for the first time, you need to download the stop words using this code:

```
nltk. download("stopwords")
```

After completion of downloading, you can load the package of stopwords from the **nltk. corpus** and use it to load the stop words.

Now, let's try to print the List of Stopwords in the English language:

```
from nltk.corpus import stopwords

#List of stopwords
stopwords = stopwords.words("english")
print(stopwords)
```

After running the above code, we get the output as:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'y
ourself', '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', 'a
n', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'b
etween', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'ar
en', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "have
n't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

We would not want these words taking up space in our database, or taking up the valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK in python has a list of stopwords stored in 16 different languages, which means you can also work with other languages also.

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

text = "India is my country"
text_tokens = word_tokenize(text)

tokens_without_sw = [word for word in text_tokens if not word in stopwords.words()]

print(tokens_without_sw)
```

```
['India', 'country']
```

You can see that the words "is" and "my" have been removed from the sentence.

Homework Problem

Here, we will discuss only one technique of noise removal, but you can try some other techniques also by yourself which we discussed in the above examples. We will discuss all those topics while we implement the NLP project.

Data Visualization for Text Data

To visualize text data, generally, we use the word cloud but there are some other techniques also, which we can try to visualize the data such as,

- Frequency Distribution

Part 3: Step by Step Guide to NLP – Text Cleaning and Preprocessing

Let's discuss the word cloud in a more detailed manner:

Word Cloud

Word Cloud is a data visualization technique in which words from a given text display on the main chart. Some properties associated with this chart are as follows:

- In this technique, more frequent or essential words display in a larger and bolder font,
- While less frequent or essential words display in smaller or thinner fonts.

This data visualization technique gives us a glance at what text should be analyzed, so it is a very beneficial technique in NLP tasks.

For more information, check the given documentation: WordCloud

Here, to draw the word cloud, we use the following text:

```
<class 'str'>

Once upon a time there was an old mother pig who had three little pigs and not enough food to feed them. So when they were old
enough, she sent them out into the world to seek their fortunes.

The first little pig was very lazy. He didn't want to work at all and he built his house out of straw. The second little pig wo
rked a little bit harder but he was somewhat lazy too and he built his house out of sticks. Then, they sang and danced and play
ed together the rest of the day.

The third little pig worked hard all day and built his house with bricks. It was a sturdy house complete with a fine fireplace
and chimney. It looked like it could withstand the strongest winds.

675
```

Code to print the word cloud:

```python
#Library to form wordcloud :
from wordcloud import WordCloud

#Library to plot the wordcloud :
import matplotlib.pyplot as plt

#Generating the wordcloud :
wordcloud = WordCloud().generate(text)

#Plot the wordcloud :
plt.figure(figsize = (12, 12))
plt.imshow(wordcloud)

#To remove the axis value :
plt.axis("off")
plt.show()
```

A word cloud formed is shown below:

Conclusions inferred from the above word cloud:

- The most frequent words display in larger fonts.

Part 3: Step by Step Guide to NLP – Text Cleaning and Preprocessing

Homework Problem

You have to make the above word cloud in the form of a circle.

Advantages of Word Cloud

- They are fast.
- They are engaging.
- They are simple to understand.
- They are casual and visually appealing.

Disadvantages of Word Cloud

- They are non-perfect for non-clean data.
- They lack the context of words.

Parts of Speech (POS) Tagging

Probably, in your childhood or in your High School, you may have heard the term Part of Speech (POS). It can really take a good amount of time to get the idea of what adjectives and adverbs actually are and understand the exact difference?

If we think about building a system where we can encode all this knowledge, then it may look like a very easy task, but for many decades, coding this knowledge into a machine learning model was a very hard NLP problem. POS tagging algorithms can predict the POS of the given word with a higher degree of precision.

Part of speech tags is the properties of words that define their main context, their function, and their usage in a sentence. It is defined by the relations of words with the other words in the sentence.

In this, our aim is to guess the part of speech for each token— whether it is a noun, a verb, an adjective, and so on. After knowing about the role of each word in the sentence, we will start to find out what the sentence is talking about.

Image Source: Google Images

Some of the commonly used parts of speech tags are:

- Nouns, which define any object or entity
- Verbs, which define some action and
- Adjectives or Adverbs, which act as modifiers, quantifiers, or intensifiers in any sentence.

In a sentence, every word will be associated with a proper part of the speech tag.

For Example, Let's consider the following sentence:

Part 3: Step by Step Guide to NLP – Text Cleaning and Preprocessing

In the below sentence, every word is associated with a part of the speech tag which defines its functions.

In this case, the associated POS tags with the above sentence are as follows:

- "David' has an NNP tag which means it is a proper noun,
- "has" and "purchased" belongs to verb indicating that they are the actions and
- "laptop" and "Apple store" are the nouns,
- "new" is the adjective whose role is to modify the context of the laptop.

Machine learning models or rule-based models are applied to obtain the part of speech tags of a word. The most commonly used part of speech tagging notations is provided by the Penn Part of Speech Tagging.

You can get the POS of individual words as a tuple.

```
from nltk import word_tokenize, pos_tag
s = "The name of our country is India"
print(pos_tag(word_tokenize(s)))
```

```
[('The', 'DT'), ('name', 'NN'), ('of', 'IN'), ('our', 'PRP$'), ('country', 'NN'), ('is', 'VBZ'), ('India', 'NNP')]
```

Why do we need Part of Speech (POS)?

Let's discuss the need for POS tags with the following example:

```
Sentence: Can you help me with the can?
```

Parts of speech (POS) tagging are crucial for syntactic and semantic analysis.

Therefore, for something like

In the sentence above, the word "can" has several semantic meanings.

- The first "can" is used for question formation.
- The second "can" at the end of the sentence is used to represent a container.
- The first "can" is a verb, and the second "can" is a noun.

Therefore, giving the word a specific meaning allows the program to handle it correctly in both semantic and syntactic analysis.

Applications of POS Tagging in NLP

Part of Speech tagging is used for many important purposes in NLP:

Word Sense Disambiguation

Some language words have multiple meanings according to their usage.

For Example, Consider the two sentences given below:

```
Please book my flight for Jodhpur
I am going to read this book in the flight
```

In the above sentences, the word "Book" is used in different contexts, however, the part of speech tag for both of the cases are different.

Part 3: Step by Step Guide to NLP – Text Cleaning and Preprocessing

we can also use Lesk Algorithm)

Improving Word-based Features

A learning model could learn different contexts of a word when used word as the features, however, if the part of speech tag is linked with them, the context is preserved, thus making strong features.

For Example: Consider the following sentence:

```
Sentence: "book my flight, I will read this book”
```

The tokens generated from the above text are as follows:

```
(“book”, 2), (“my”, 1), (“flight”, 1), (“I”, 1), (“will”, 1), (“read”, 1), (“this”, 1)
```

The tokens generated from the above text with POS are as follows:

```
(“book_VB”, 1), (“my_PRP$”, 1), (“flight_NN”, 1), (“I_PRP”, 1), (“will_MD”, 1), (“read_VB”, 1), (“this_DT”,
1), (“book_NN”, 1)
```

Normalization and Lemmatization

POS tags are the basis of the lemmatization process for converting a word to its base form (lemma).

Efficient Stopword Removal

POS tags are also useful in the efficient removal of stopwords.

For Example, there are some tags that always define the low frequency / less important words of a language. For example: (IN – “within”, “upon”, “except”), (CD – “one”,” two”, “hundred”), (MD – “may”, “must” etc)

This ends our Part-3 of the Blog Series on Natural Language Processing!

End Notes

Thanks for reading!

If you liked this and want to know more, go visit my other articles on Data Science and Machine Learning by clicking on the Link

Please feel free to contact me on Linkedin, Email.

Something not mentioned or want to share your thoughts? Feel free to comment below And I’ll get back to you.

About the Author

Chirag Goyal

Currently, I am pursuing my Bachelor of Technology (B.Tech) in Computer Science and Engineering from the **Indian Institute of Technology Jodhpur(IITJ).** I am very enthusiastic about Machine learning, Deep Learning, and Artificial Intelligence.

The media shown in this article are not owned by Analytics Vidhya and are used at the Author’s discretion.

blogathon Natural Language Processing Python

Part 3: Step by Step Guide to NLP – Text Cleaning and Preprocessing

About the Author

Chirag Goyal

Our Top Authors

Download

Analytics Vidhya App for the Latest blog/Article

Previous Post

Part 2: Step by Step Guide to NLP – Knowledge Required to Learn NLP

Next Post

Dimensionality Reduction using AutoEncoders in Python

Leave a Reply

Your email address will not be published. Required fields are marked *

Part 3: Step by Step Guide to NLP – Text Cleaning and Preprocessing

Name*

Email*

Website

Notify me of follow-up comments by email.

Notify me of new posts by email.

Submit

Top Resources

Python Tutorial: Working with CSV file for Data Science

Harika Bonthu - AUG 21, 2021

Realistic Face Restoration with GFP-GAN and DFDNet

Suvojit Hore - MAY 27, 2022

Understanding Support Vector Machine(SVM) algorithm from examples (along with code)

Sunil Ray - SEP 13, 2017

Boost Model Accuracy of Imbalanced COVID-19 Mortality Prediction Using GAN-based..

Bala Gangadhar Thilak Adiboina - OCT 07, 2020

Analytics Vidhya

Analytics Vidhya

About Us

Our Team

Careers

Data Scientists

Blog

Hackathon

Discussions

Part 3: Step by Step Guide to NLP – Text Cleaning and Preprocessing

Companies

Post Jobs

Trainings

Hiring Hackathons

Advertising

Visit us

© Copyright 2013-2022 Analytics Vidhya.

Privacy Policy | Terms of Use | Refund Policy