Text Preprocessing in NLP with Python Codes

DATA SCIENCE INTERMEDIATE MACHINE LEARNING NLP PYTHON TEXT UNSTRUCTURED DATA

Text preprocessing is an essential step in natural language processing (NLP) that involves cleaning and transforming unstructured text data to prepare it for analysis. It includes tokenization, stemming, lemmatization, stop-word removal, and part-of-speech tagging. In this article, we will introduce the basics of text preprocessing and provide Python code examples to illustrate how to implement these tasks using the NLTK library. By the end of the article, readers will better understand how to prepare text data for NLP tasks.

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

Table of contents

What is Text Preprocessing in NLP?

Natural Language Processing (NLP) is a branch of Data Science which deals with Text data. Apart from numerical data, Text data is available to a great extent which is used to analyze and solve business problems. But before using the data for analysis or prediction, processing the data is important.

To prepare the text data for the model building we perform text preprocessing. It is the very first step of NLP projects. Some of the preprocessing steps are:

- Removing punctuations like . , ! $( ) * % @
- Removing URLs
- Removing Stop words
- Lower casing
- Tokenization
- Stemming
- Lemmatization

Why is Text Preprocessing important?

Text preprocessing is crucial in natural language processing (NLP) for several reasons:

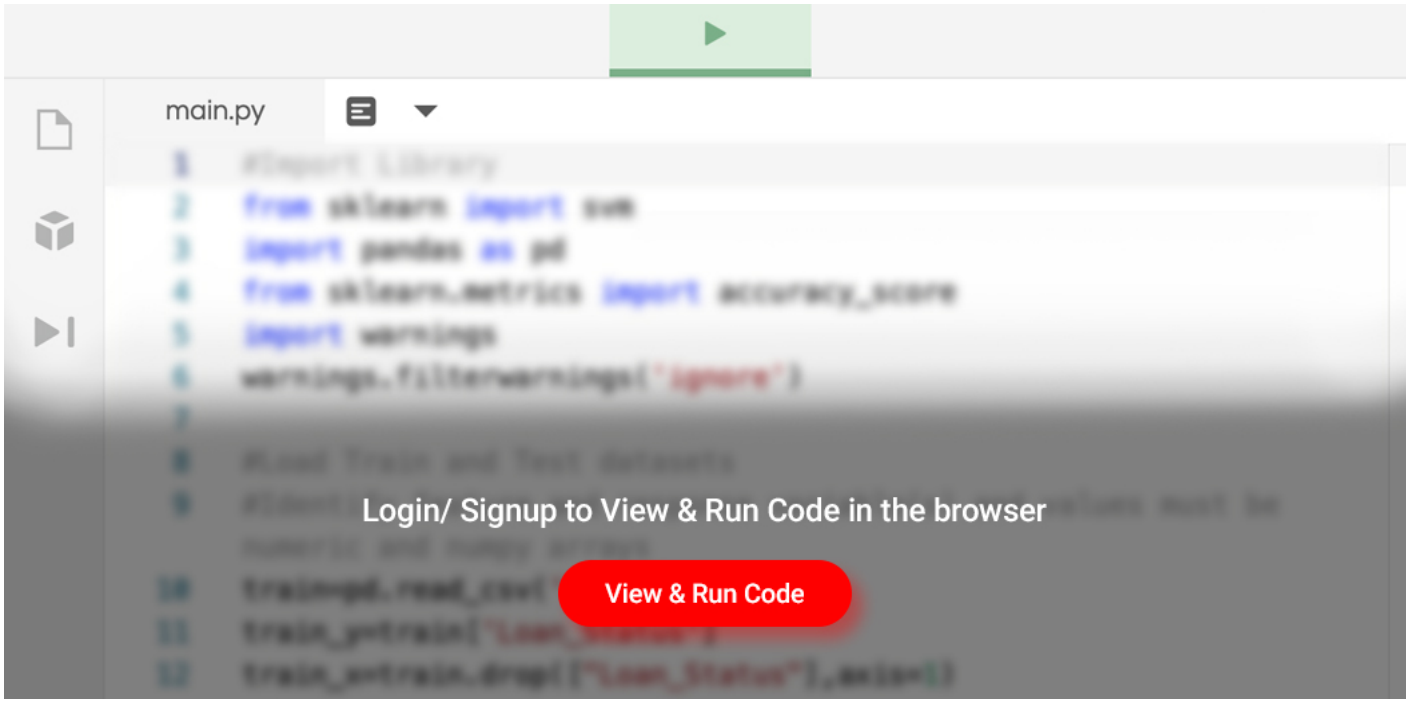
Preprocessing Task	Reasons
Noise Reduction	Text data often contains noise such as punctuation, special characters, and irrelevant symbols. Preprocessing helps remove these elements, making the text cleaner and easier to analyze.
Normalization	Different forms of words (e.g., “run,” “running,” “ran”) can convey the same meaning but appear in different forms. Preprocessing techniques like stemming and lemmatization help standardize these variations.
Tokenization	Text data needs to be broken down into smaller units, such as words or phrases, for analysis. Tokenization divides text into meaningful units, facilitating subsequent processing steps like feature extraction.
Stopword Removal	Stopwords are common words like “the,” “is,” and “and” that often occur frequently but convey little semantic meaning. Removing stopwords can improve the efficiency of text analysis by reducing noise.
Feature Extraction	Preprocessing can involve extracting features from text, such as word frequencies, n-grams, or word embeddings, which are essential for building machine learning models.
Dimensionality Reduction	Text data often has a high dimensionality due to the presence of a large vocabulary. Preprocessing techniques like term frequency-inverse document frequency (TF-IDF) or dimensionality reduction methods can help.

Overall, text preprocessing plays a crucial role in preparing text data for NLP tasks by improving data quality, reducing noise, and facilitating effective analysis and modeling.

SMS Spam Data for Text Preprocessing

We need to use the required steps based on our dataset. In this article, we will use **SMS Spam data** to understand the steps involved in Text Preprocessing in NLP.

Let’s start by importing the pandas library and reading the data.



	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
#expanding the dispay of text sms column pd.set_option('display.max_colwidth', -1) #using only v1 and v2 column data= data [['v1','v2']] data.head()
```

	v1	v2
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives around here though

The data has 5572 rows and 2 columns. You can check the shape of data using data.shape function. Let's check the dependent variable distribution between spam and ham.

```
#checking the count of the dependent variable data['v1'].value_counts()
```

```
ham     4825
spam    747
Name: v1, dtype: int64
```

Steps to Clean the Data

Punctuation Removal

In this step, all the punctuations from the text are removed. string library of Python contains some pre-defined list of punctuations such as '!"#$%&'()*+,-./:;?@[\]^_`{|}~'

```
#library that contains punctuation import string string.punctuation
```

```
#defining the function to remove punctuation def remove_punctuation(text): punctuationfree="".join([i for i in text if i not in string.punctuation]) return punctuationfree #storing the puntuation free text data['clean_msg']= data['v2'].apply(lambda x:remove_punctuation(x)) data.head()
```

v2	clean_msg
Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	Go until jurong point crazy Available only in bugis n great world la e buffet Cine there got amore wa
Ok lar... Joking wif u oni...	Ok lar Joking wif u on
Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005 Text FA t 87121 to receive entry questionstd txt rateTCs apply 08452810075over18
U dun say so early hor... U c already then say...	U dun say so early hor U c already then sa
Nah I don't think he goes to usf, he lives around here though	Nah I dont think he goes to usf he lives around here though

We can see in the above output, all the punctuations are removed from v2 and stored in the clean_msg column.

Lowering the Text

It is one of the most common text preprocessing Python steps where the text is converted into the same case preferably lower case. But it is not necessary to do this step every time you are working on an NLP problem as for some problems lower casing can lead to loss of information.

For example, if in any project we are dealing with the emotions of a person, then the words written in upper cases can be a sign of frustration or excitement.

```
data['msg_lower']= data['clean_msg'].apply(lambda x: x.lower())
```

Output: All the text of clean_msg column are converted into lower case and stored in msg_lower column

clean_msg	msg_lower
Go until jurong point crazy Available only in bugis n great world la e buffet Cine there got amore wat	go until jurong point crazy available only in bugis n great world la e buffet cine there got amore wat
Ok lar Joking wif u oni	ok lar joking wif u oni
Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005 Text FA to 87121 to receive entry questionstd txt rateTCs apply 08452810075over18s	free entry in 2 a wkly comp to win fa cup final tkts 21st may 2005 text fa to 87121 to receive entry questionstd txt ratetcs apply 08452810075over18s
U dun say so early hor U c already then say	u dun say so early hor u c already then say
Nah I dont think he goes to usf he lives around here though	nah i dont think he goes to usf he lives around here though

Tokenization

In this step, the text is split into smaller units. We can use either sentence tokenization or word tokenization based on our problem statement.

```
#defining function for tokenization import re def tokenization(text): tokens = re.split('W+',text) return tokens #applying function to the column data['msg_tokenied']= data['msg_lower'].apply(lambda x: tokenization(x))
```

Output: Sentences are tokenized into words.

msg_lower	msg_tokenied
go until jurong point crazy available only in bugis n great world la e buffet cine there got amore wat	[go, until, jurong, point, crazy, available, only, in, bugis, n, great, world, la, e, buffet, cine, there, got, amore, wat]
ok lar joking wif u oni	[ok, lar, joking, wif, u, oni]
free entry in 2 a wkly comp to win fa cup final tkts 21st may 2005 text fa to 87121 to receive entry questionstd txt ratetcs apply 08452810075over18s	[free, entry, in, 2, a, wkly, comp, to, win, fa, cup, final, tkts, 21st, may, 2005, text, fa, to, 87121, to, receive, entry, questionstd, txt, ratetcs, apply, 08452810075over18s]
u dun say so early hor u c already then say	[u, dun, say, so, early, hor, u, c, already, then, say]
nah i dont think he goes to usf he lives around here though	[nah, i, dont, think, he, goes, to, usf, he, lives, around, here, though]

Stop Word Removal

Stopwords are the commonly used words and are removed from the text as they do not add any value to the analysis. These words carry less or no meaning.

NLTK library consists of a list of words that are considered stopwords for the English language. Some of them are : [i, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd, your, yours, yourself, yourselves, he, most, other, some, such, no, nor, not, only, own, same, so, then, too, very, s, t, can, will, just, don, don't, should, should've, now, d, ll, m, o, re, ve, y, ain, aren't, could, couldn't, didn't, didn't]

But it is not necessary to use the provided list as stopwords as they should be chosen wisely based on the project. **For example**, 'How' can be a stop word for a model but can be important for some other problem where we are working on the queries of the customers. We can create a customized list of stop words for different problems.

```
#importing nlp library import nltk #Stop words present in the library stopwords = nltk.corpus.stopwords.words('english') stopwords[0:10] ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]


#defining the function to remove stopwords from tokenized text def remove_stopwords(text): output= [i for i in text if i not in stopwords] return output


#applying the function data['no_stopwords']= data['msg_tokenied'].apply(lambda x:remove_stopwords(x))
```

Output: Stop words that are present in the nltk library such as in, until, to, I, here are removed from the tokenized text and the rest are stored in the no_stopwords column.

msg_tokenied	no_stopwords
[go, until, jurong, point, crazy, vailable, only, in, bugis, n, great, world, la, e, buffet, cine, there, got, amore, wat]	[go, jurong, point, crazy, available, bugis, n, great, world, la, e, buffet, cine, got, amore, wat]
[ok, lar, joking, wif, u, oni]	[ok, lar, joking, wif, u, oni]
[free, entry, in, 2, a, wkly, comp, to, win, fa, cup, final, tkts, 21st, may, 2005, text, fa, to, 87121, to, receive, entry, questionstd, txt, ratetcs, apply, 08452810075over18s]	[free, entry, 2, wkly, comp, win, fa, cup, final, tkts, 21st, may, 2005, text, fa, 87121, receive, entry, questionstd, txt, ratetcs, apply, 08452810075over18s]
[u, dun, say, so, early, hor, u, c, already, then, say]	[u, dun, say, early, hor, u, c, already, say]
[nah, i, dont, think, he, goes, to, usf, he, lives, around, here, though]	[nah, dont, think, goes, usf, lives, around, though]

Stemming

It is also known as the text standardization step where the words are stemmed or diminished to their root/base form. **For example**, words like 'programmer', 'programming, 'program' will be stemmed to 'program'.

But the **disadvantage** of stemming is that it stems the words such that its root form loses the meaning or it is not diminished to a proper English word. We will see this in the steps done below.

```
#importing the Stemming function from nltk library from nltk.stem.porter import PorterStemmer #defining the object for stemming porter_stemmer = PorterStemmer()


#defining a function for stemming def stemming(text): stem_text = [porter_stemmer.stem(word) for word in text] return stem_text data['msg_stemmed']=data['no_sw_msg'].apply(lambda x: stemming(x))
```

Output: In the below image, we can see how some words are stemmed to their base.

crazy-> crazi

available-> avail

entry-> entri

early-> earli

no_stopwords	msg_stemmed
[go, jurong, point, crazy, available, bugis, n, great, world, la, e, buffet, cine, got, amore, wat]	[go, jurong, point, crazi, avail, bugi, n, great, world, la, e, buffet, cine, got, amor, wat]
[ok, lar, joking, wif, u, oni]	[ok, lar, joke, wif, u, oni]
[free, entry, 2, wkly, comp, win, fa, cup, final, tkts, 21st, may, 2005, text, fa, 87121, receive, entry, questionstd, txt, ratetcs, apply, 08452810075over18s]	[free, entri, 2, wkli, comp, win, fa, cup, final, tkt, 21st, may, 2005, text, fa, 87121, receiv, entri, questionstd, txt, ratetc, appli, 08452810075over18]
[u, dun, say, early, hor, u, c, already, say]	[u, dun, say, earli, hor, u, c, alreadi, say]
[nah, dont, think, goes, usf, lives, around, though]	[nah, dont, think, goe, usf, live, around, though]

Now let's see how Lemmatization is different from Stemming.

Lemmatization

It stems the word but makes sure that it does not lose its meaning. Lemmatization has a pre-defined dictionary that stores the context of words and checks the word in the dictionary while diminishing.

The difference between Stemming and Lemmatization can be understood with the example provided below.

Original Word	After Stemming	After Lemmatization
goose	goos	goose
geese	gees	goose

```
from nltk.stem import WordNetLemmatizer #defining the object for Lemmatization wordnet_lemmatizer = WordNetLemmatizer()
```

```
#defining the function for lemmatization def lemmatizer(text): lemm_text = [wordnet_lemmatizer.lemmatize(word) for word in text] return lemm_text
data['msg_lemmatized']=data['no_stopwords'].apply(lambda x:lemmatizer(x))
```

Output: The difference between Stemming and Lemmatization can be seen in the below output.

In the first row- crazy has been changed to **crazi** which has no meaning but for lemmatization, it remained the same i.e **crazy**

In the last row- goes has changed to **goe** while stemming but for lemmatization, it has converted into **go** which is meaningful.

no_stopwords	msg_stemmed	msg_lemmatized
[go, jurong, point, crazy, available, bugis, n, great, world, la, e, buffet, cine, got, amore, wat]	[go, jurong, point, crazi, avail, bugi, n, great, world, la, e, buffet, cine, got, amor, wat]	[go, jurong, point, crazy, available, bugis, n, great, world, la, e, buffet, cine, got, amore, wat]
[ok, lar, joking, wif, u, oni]	[ok, lar, joke, wif, u, oni]	[ok, lar, joking, wif, u, oni]
[free, entry, 2, wkly, comp, win, fa, cup, final, tkts, 21st, may, 2005, text, fa, 87121, receive, entry, questionstd, txt, ratetcs, apply, 08452810075over18s]	[free, entri, 2, wkli, comp, win, fa, cup, final, tkt, 21st, may, 2005, text, fa, 87121, receiv, entri, questionstd, txt, ratetc, appli, 08452810075over18]	[free, entry, 2, wkly, comp, win, fa, cup, final, tkts, 21st, may, 2005, text, fa, 87121, receive, entry, questionstd, txt, ratetcs, apply, 08452810075over18s]
[u, dun, say, early, hor, u, c, already, say]	[u, dun, say, earli, hor, u, c, alreadi, say]	[u, dun, say, early, hor, u, c, already, say]
[nah, dont, think, goes, usf, lives, around, though]	[nah, dont, think, goe, usf, live, around, though]	[nah, dont, think, go, usf, life, around, though]

After all the text processing steps are performed, the final acquired data is converted into the numeric form using Bag of words or TF-IDF.

Endnotes

Apart from the steps shown in this article, many other steps are a part of preprocessing. Some of them are URL removal, HTML tags removal, Rare words removal, Frequent words removal, Spelling checking, and many more. The steps must be chosen based on the dataset you are working on and what is necessary for the project.

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

Article Url - https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/

Deepanshi