



Search Medium

Write



# Preprocessing Text Data For NLP Models

A tutorial on various text data manipulation methods for NLP data processing.



Josephine Amponsah · Following

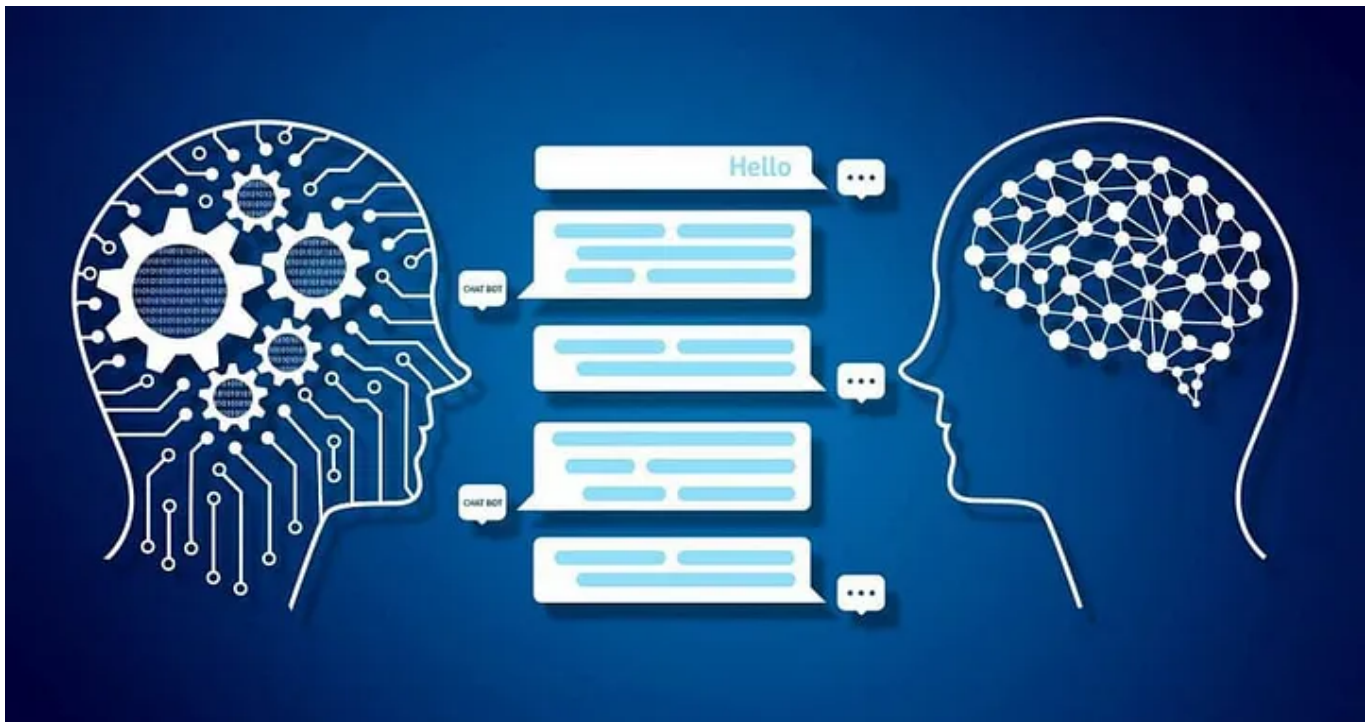
8 min read · Jun 9



55



1



Natural Language Processing(NLP) is a rapidly growing space in AI/ML and it's now widely used by various entities. As a data scientist, knowing the right

preprocessing methods to perform on data before NLP tasks are essential to performance of your models. In this article, we will explore all the preprocessing methods and when each apply.

## Pre-requisites

- Understand computation of vectors and matrices
- Experience with regex, and data cleaning with pandas

## NLP Glossary

1. **Document:** refers to any form of text data such as articles or even a sentence and as units of data when used for NLP.
2. **Corpora:** refers to a collection of documents , often serving as datasets for NLP research or model training. Examples are google reviews of a place, blog posts by several media outlets, etc.
3. **Tokens:** refers to the basic units of a document as used in an NLP task. These could be individual words, pairs, or any grouping of words as influenced by the purpose of document.
4. **Syntax:** defines the grammatical structures of a language. It guides principles that define how words can be put together to form sentences or phrases.
5. **Semantics:** defines how to put words together so that they actually make sense when organized based on the available syntactical rules.
6. **Word embeddings:** a learned representation of a word wherein each word is represented using a vector in n-dimensional space.

## Handling Text Data

- **Tokenization**

As a fundamental step in text processing, tokenization is a segmentation technique that breaks down a document to meaningful tokens. The output can be either tokens of words, numbers or expressions containing emoticons and is determined by the document structure and method used.

```
# retrieving first comment
comment = "I am excited about my career as a Data Scientist"

# splitting text to tokens
tokenized_comment = word_tokenize(comment)

#output
['I', 'am', 'excited', 'about', 'my', 'career', 'as', 'a', 'Data', 'Scientist']
```

### *Types of Tokenizers :*

- *Regular expressions-based tokenizer*

This tokenizer allows for segmentation based on text patterns with the use of regex.

```
# importing regex tokenizer
from nltk.tokenize import RegexpTokenizer
review = "The rooms, priced at GHS1000, wasn't worth the value I paid"

#regex pattern for text and numerical combinations
pattern = '\w+|\GHS[\d\.]+\S+'
```

```
tokenizer = Regexp(pattern)
tokens = tokenizer.tokenize(review)
```

- *Treebank tokenizer*

The treebank tokenizer, which also accepts regex patterns is most beneficial when handling contracted negated texts such as *doesn't, I'm, we'll*. This simplifies the extraction of the stem words, which will be discussed under Stemming.

- *Tweet tokenizer*

In the rise of social media, it is essential that informal language pertaining to space is considered, when segmenting documents obtained from such platforms. Tweet tokenizer has been built to understand patterns such as repetition of alphabets, emojis and hashtag. This helps provide better translation of the context and sentiment surrounding a text document.

```
#import TweetTokenizer class
from nltk.tokenize import TweetTokenizer
tweet = '@lady__jossy i love thissss 😍 #romantic #vacation #excited'

#using attributes to strip handles and manage stretched words
tokenizer = TweetTokenizer(strip_handles=True, reduce_len=True)
tweet_tokens = tokenizer.tokenize(tweet)
```

- *N-grams*

In the above tokenization techniques, the resulting tokens are singular words. However, there are several compound words such as names of people, places or even things that lose their meaning if separated. N-gram allows the specification of the number of words that tokens are

broken down to, making room for such cases and hence, preserving the meaning of compound words occurring in a document.

They are called bigrams if number specified is 2, trigrams if 3 , etc.

```
#import ngrams
from nltk.util import ngrams

text = "Abu Dhabi is as beautiful a city as Dubai"
tokens = text.split()
bigrams = list(ngrams(tokens, 2))
```

- **Stopword Removal**

Stopwords are frequently occurring words that are required to make sentences grammatical accurate but do not hold carry weight in context of a document. These are often articles (the, an, a ), pronouns (she, they, he), forms of *to be* (will, is, shall), etc. Removing stopwords reduces the size of your vocabulary, making word searches more efficient. The nltk library has a store of stopwords, which can be modified depending on the corpora being handled.

```
#downloading stopwords from nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

#creating a set of english stopwords
stop = set(stopwords.words('english'))
filtered_tokens = []

#removing stopwords to output of tokenization example
for word in tokenized_comment:
    if word not in stop:
        filtered_tokens.append(word)
```

```
#output  
['I', 'excited', 'career', 'Data', 'Scientist']
```

*If you are encountering errors downloading the stopwords, run 'nltk.download()' to download all data in the nltk library first before you continue.*

- **Stemming**

Stemming is a word normalization technique that converts words to their root forms. For instance, the words *sanitization* and *sanitizer* come from the word *sanitize*, which is known as the stem. The conversion is done by removing affixes from words. This process helps reduce size of vocabulary, reduce search time and improve feature representation.

There are two common types of stemmers: *Porter* and *Snowball* stemmer. The difference between the two types is that, Porter stemmer supports only english documents while, Snowball supports multiple languages.

```
#importing stemmer  
from nltk.stem.porter import PorterStemmer  
  
stemmer = PorterStemmer()  
stems = [stemmer.stem(word) for word in filtered_tokens]  
  
#output  
['i', 'excit', 'career', 'data', 'scientist']
```

- **Lemmatization**

It is a word normalization technique, which takes into account the neighbouring words for context, part-of-speech tags (noun, verb, adjective, etc) and meaning of the words. *WordNet*, *Spacy*, *TextBlob*, *Gensim* are all types of lemmatizers. In the following example, we will use the *TextBlob Lemmatizer*.

```
# ensure to install install textblob with the command "pip install textblob"
from textblob import TextBlob, Word

lemma_tokens = [Word(word).lemmatize() for word in filtered_tokens]

#output
['I', 'excited', 'career', 'Data', 'Scientist']
```

With a look at the output of the *stemmer*, that the word **‘excited’** is stripped of its suffix **‘ed’**. In the output of the *lemmatizer* however, the word stays as is, showing that when the semantics is considered, the word is needed in its current form.

- **Part-of-Speech (POS) Tagging**

POS tagging is method used to identify the part-of-speech of a word as it appears in a sentence. This helps to provide the contextual meaning of words, as some words have different meanings depending how they are used in a sentence.

Take a look at the word *answer* following sentences:

1. He gave an *answer* to the question.
2. They *answer* to the manager.

In sentence 1, the word answer is a noun, but used as a verb in sentence 2. This piece of information, is hence critical to any interpretation for any NLP model.

```
from textblob import Textblob

sent1 = TextBlob("He gave an answer to the question")
sent2 = TextBlob("They answer to the manager")

sent1_tags = sent1.tags
sent2_tags = sent2.tags

#output
#sent1_tags
[('They', 'PRP'), ('answer', 'VBP'), ('to', 'TO'), ('the', 'DT'),
 ('manager', 'NN')]

#sent2_tags
[('He', 'PRP'), ('gave', 'VBD'), ('an', 'DT'), ('answer', 'NN'), ('to', 'TO'),
 ('the', 'DT'), ('question', 'NN')]

# view the definition on POS tags with the following line of code

nltk.help.upenn_tagset()
```

## Text to Numbers Transformation

In this section, we will use the following preprocessed data to run examples.

```
def remove_stopwords(tokens):
    filtered = []
    for word in tokens:
```



```
        if word not in stop:
            filtered.append(word)

    return filtered

data = ["Natural Language Processing is a very useful machine learning technique",
        "Learning machine learning concepts can be very confusing as an aspiring",
        "Data Scientists are doing magic with Natural Language Processing",
        "It's been fun exploring the world of Artificial Intelligence and Machine Learning"]

def preprocess(sent_list):
    sentences = []
    for sent in data:
        line = word_tokenize(sent)
        new_line = remove_stopwords(line)
        new_sent = " ".join(new_line)
        sentences.append(new_sent)
    return sentences

processed_data = preprocess(data)
```

- **Count Vectorizer**

Count Vectorizer is a method for converting text data to numerical data in the form of tokens, based on the Bag of Words(BoW) technique.

BoW is a representation of text, describing the occurrence of words within a document. Based on a vocabulary built from the corpus, each word and its occurrence in each document form key-value pairs.

### *Technical Implementation*

1. Create a list of the documents (in this case reviews)
2. Tokenize data
3. Remove stop words
4. Perform any other text preprocessing method you deem fit

*Assuming preprocessed data is named 'processed\_reviews'.*

```
from sklearn.feature_extraction.text import CountVectorizer
# create an instance of the vectorizer
vectorizer = CountVectorizer()
# run model on preprocessed corpus
cv_matrix = vectorizer.fit_transform(preprocessed_reviews)
#view vocabulary
vocab = vectorizer.get_feature_names_out()

#view matrix
matrix = cv_matrix.toarray()

#output
#vocab
array(['artificial', 'aspiring', 'concepts', 'confusing', 'data',
       'exploring', 'fun', 'intelligence', 'it', 'language', 'learning',
       'machine', 'magic', 'natural', 'processing', 'scientist',
       'scientists', 'technique', 'useful', 'world'], dtype=object)

#matrix
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0],
       [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1]])
```

*Parameters such as the minimum and maximum occurrence of a word across document, n-gram range can be passed as attributes to the Count vectorizer. This helps reduce the size of vocabulary among other benefits.*

- **Term frequency- inverse document frequency (TF-IDF)**

*TF-IDF* uses the same approach as Count Vectorizer to generate document vectors, based on its frequency. However, it accounts for words that may valuable information in a document but occurs less frequent in other documents in the corpus.

It uses a technique known as *Inverse Document Frequency*, which suppresses the weight of frequently occurring words while scaling up the weight of meaningful yet less occurring terms.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()

tf_idf_matrix = vectorizer.fit_transform(preprocessed_corpus)
matrix = tf_idf_matrix.toarray()

#output
array([[0.          , 0.          , 0.          , 0.          , 0.          ,
        0.          , 0.          , 0.          , 0.          , 0.36445896,
        0.29506094, 0.29506094, 0.          , 0.36445896, 0.36445896,
        0.          , 0.          , 0.46226998, 0.46226998, 0.          ],
       [0.          , 0.38753157, 0.38753157, 0.38753157, 0.30553434,
        0.          , 0.          , 0.          , 0.          , 0.          ,
        0.49471276, 0.24735638, 0.          , 0.          , 0.          ,
        0.38753157, 0.          , 0.          , 0.          , 0.          ],
       [0.          , 0.          , 0.          , 0.          , 0.37222485,
        0.          , 0.          , 0.          , 0.          , 0.37222485,
        0.          , 0.          , 0.47212003, 0.37222485, 0.37222485,
        0.          , 0.47212003, 0.          , 0.          , 0.          ],
       [0.38306527, 0.          , 0.          , 0.          , 0.          ,
        0.38306527, 0.38306527, 0.38306527, 0.38306527, 0.          ,
        0.2445056 , 0.2445056 , 0.          , 0.          , 0.          ,
        0.          , 0.          , 0.          , 0.          , 0.38306527]])
```

- **Word2Vec**

It is an unsupervised methodology for building word embeddings, where the semantic and syntactic relationship of words are both considered. Unlike *Count Vectorizer* and *TF-IDF*, the context of the word influences the word embedding, hence enabling the identification of synonyms and antonyms among other word relationships.

Google has a pre-trained model in the [gensim](#) library, built on google news datasets. However, it is essential to train the model with parts of the corpus the model will be used on. This allows the model to learn the vocabulary and context of the specific branch of knowledge.

```
# for Word2Vec, each sentence has to be a list of tokenized word
input = []
for sent in processed_data:
    sent_token = word_tokenize(sent)
    input.append(sent_token)

# word2vec with pretrained models
import gensim
from gensim.models import Word2Vec

# train model on part of corpus
model = Word2Vec(input, min_count = 2)

# resulting matrix
matrix = model.wv.vectors
```

*The model can then be run on the entire tokenized corpus to obtain vectors of sentences, each with word vectors. This numerical data can be used for tasks such as document clustering, simplifying analysis of large bodies of text.*

## Conclusion

The text preprocessing methods used for any project, depends on the NLP task and the data being used. It is therefore essential to understand the structure of input data for a chosen NLP model, before diving into preprocessing the data.

## References

- Hands-on Natural Language Processing by *Aman Kedia, Mayank Rasu*
- Natural Language Processing Fundamentals by *Sohom Ghosh, Dwight Gunning*

*I hope you learnt something useful from this article. Kindly comment with your feedback. Subscribe to get notified of future articles.*

Naturallanguageprocessing

Machine Learning

Feature Engineering

Artificial Intelligence

Data Science

## More from the list: "NLP"

Curated by Himanshu Birla



Jon Gi... in Towards Data ...

### Characteristics of Word Embeddings

★ · 11 min read · Sep 4, 2021



Jon Gi... in Towards Data ...

### The Word2vec Hyperparameters

★ · 6 min read · Sep 3, 2021



Jon Gi... in

### The Word2vec

★ · 15 min read



[View list](#)



## Written by Josephine Amponsah

29 Followers

Data Scientist | Product Manager

Following



### More from Josephine Amponsah



Josephine Amponsah

## HIERARCHICAL FORECASTING: RETAIL INVENTORY PLANNING

In order to forecast revenue or demand of inventory, the behaviour of each product ove...

4 min read · Mar 16



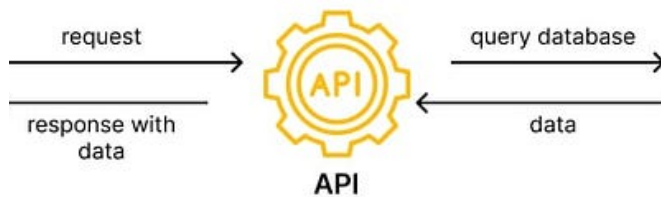
18



1



...



Josephine Amponsah

## How to Build Machine Learning Applications with FastAPI

A tutorial on building FastAPI for Machine Learning model usability

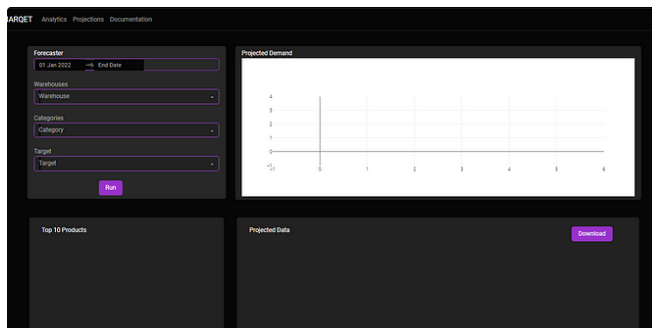
7 min read · Jun 18



22



...



## HIERARCHICAL FORECASTING: DEPLOYING MODEL TO...

4 min read · Apr 23



4



## ANALYZING IMPACT OF DIGITAL MARKETING

4 min read · Mar 5

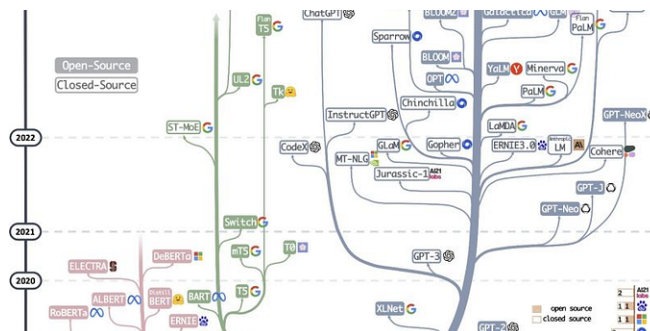


3



See all from Josephine Amponsah

## Recommended from Medium





Lukas Niederhäuser

## Exploratory Text Analysis of Swiss and German companies from...

The objective of this article is to gather and analyse text from Wikipedia pages that...

8 min read · Jul 5



39



Haifeng Li

## A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

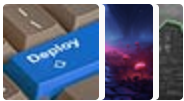
15 min read · Sep 14



372



### Lists



#### Predictive Modeling w/ Python

20 stories · 452 saves



#### Practical Guides to Machine Learning

10 stories · 519 saves



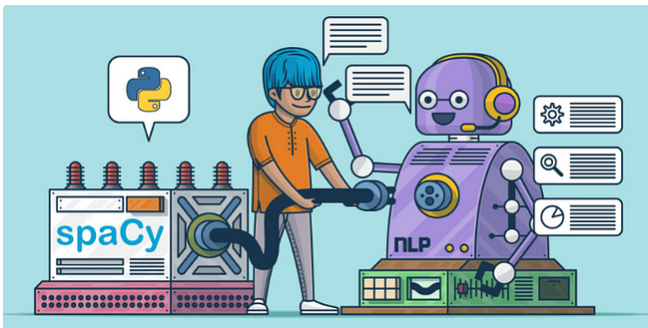
#### Natural Language Processing

669 stories · 283 saves



#### ChatGPT prompts

24 stories · 459 saves



HasancanÇakıcıoğlu

## Comprehensive Text Preprocessing NLP (Natural Language...

Text preprocessing plays a crucial role in Natural Language Processing (NLP) by...

12 min read · Jul 9



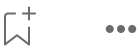
Nimrita Koul


## Natural Language Processing with Python Part 5: Text Classification

This article is the fifth in the series of my articles covering the sessions I delivered for...

8 min read · May 4



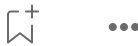


 Mustafa Germec, PhD in Python in Plain English

# Text Preprocessing with Natural Language Processing (NLP)

Mustafa Germec, PhD

20 min read · Sep 26



```
ssed his claim to be the greatest player of all time after another performan

S:
ted: {entity['word']], Entity Label: {entity['entity_group']], Confidence sco

jokovic, Entity Label: PER, Confidence score: 0.9974638223648071
Open, Entity Label: MISC, Confidence score: 0.9965554475784302
Entity Label: LOC, Confidence score: 0.9993627667427063
ntity Label: MISC, Confidence score: 0.9981368780136108
Nadal, Entity Label: PER, Confidence score: 0.9987477660179138
Entity Label: MISC, Confidence score: 0.9151148796881543
```

 Seffa B

# Named Entity Recognition with Transformers: Extracting Metadata

3 min read · Jun 12

See more recommendations