**Bugspeed courses:**

[**http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-1/**](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-1/)

[**http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-2/**](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-2/)

[**http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-3/**](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-3/)

[**http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-4/**](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-4/)

**Natural Language Processing – Beginner to Advanced (PART – 1)**

[**http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-1/**](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-1/)

[1 Comment](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-1/#comments) / [Programming](http://bugspeed.xyz/category/programming/) / By [admin](http://bugspeed.xyz/author/admin/)

Text

Description automatically generated

So the first question that comes in our mind is **What is NLP** ? Why is it so important and so much famous these days.

To understand it’s importance, let’s look at some promising examples :-

So have you ever wondered when we are using famous messaging apps like Whats App, Messenger, Hike etc, they suggest meaningful words before letting you complete the sentence. Another example would be like the SPAM or junk folder in your email, Chat Bots, Google Translation and so much more !  
So yeah, these were some cool examples of NLP or Natural Language Processing.

So the term Natural Language Processing can be defined as field concerned with the ability of a computer to understand, analyze, manipulate and potentially generate human language (or close to human language).  
It can be any language English, Hindi, French, Spanish etc.

**Real Life Examples** :-

* Auto – Complete
* Auto – Correct
* Spam Detection
* Translation of one Language to Another
* Conversational Chat Bots

**Areas of NLP :-**

* **Sentiment Analysis** :-  
  *Sentiment Analysis* is a natural language processing technique used to determine whether data is positive, negative or neutral. Sentiment analysis is often performed on textual data to help businesses monitor brand and product sentiment in customer feedback, and understand customer needs.  
  For example :- Movie reviews sentiment analysis, tweets analysis etc.
* **Topic Modeling** :-   
  A *topic model* is one that automatically discovers topics occurring in a collection of documents. A trained model may then be used to discern which of these topics occur in new documents. The model can also pick out which portions of a document cover which topics.
* **Text Classification** :-  
  Text clarification is the process of categorizing the text into a group of words. By using NLP, text classification can automatically analyze text and then assign a set of predefined tags or categories based on its context.

Now that we have some clue about what’s going on and what is Natural Language Processing, we will continue the *NLP WITH PYTHON* part…

**NLP ToolKit – NLTK**

NLTK i.e. Natural Language Processing Tool Kit is a suite of open-source tools created to make NLP processes in Python easier to build.   
  
In the above lesson we have seen that how NLP has revolutionized many areas of language such as sentiment analysis, part-of-speech tagging, text classification, language translation, topic modeling, language generation and many many more. So there are many in-built functions and libraries that are included inside this NLTK library.

!pip install nltk

import nltk

nltk.download()

# this will allow you to download all the necessary tools present in the library

This library let us do all the necessary preprocessing on our text data without any pain :D, some of the components of this library are :- **stemming**,**lematizing***,* **tokenizing***, stop-words removal* and so many more…  
  
*We will discuss all of these functions and various useful techniques in my next part…  
So don’t you worry folks ! I got you all covered :D.   
Happy Learning !*

[Click here](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-2/) for part – 2 of this series.

<http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-2/>

In the part 2 of this series, we will talk about NLP pipepline and various important techniques which are included and implemented in NLTK Python library, so without wasting any time, let’s get straight to it !  
  
If you have not read my previous blog, then do check it out ! [Click here](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-1/) for part – 1.

### NLP Pipeline :-

Whenever we work on raw text data, python does not understand words, it just sees a stream of characters and for it, all the characters are same having no meaning. Any Machine Learning algorithm/programming language only understands **Numbers**/**Vectors** and not words so in order to make it understand, we need to perform the above NLP pipeline as shown in the image above.

#### Tokenization

Tokenization is simply splitting the text/corpus into words or sentences.

from nltk.tokenize import word\_tokenize

sample\_text = 'Hi my name is Yash'

tokenized\_text = word\_tokenize(sample\_text)

print(tokenized\_text)

output : [‘Hi’, ‘my’, ‘name’, ‘is’, ‘Yash’]

#### Text Cleaning

Text cleaning basically refers to the functions applied to the raw text in order to remove unnecessary words, punctuation, extra white spaces, and giving the text more meaning in order to be processed by our ML algorithm.  
There are various Text Pre-processing/Cleaning techniques like :

* Punctuation Removal
* Stop Words Removal
* Extra white space Removal
* Emoji Removal
* Emoticons Removal
* HTML tags Removal
* URLs Removal
* Conversion to Lower case
* Numbers Removal
* Expanding Contractions and so many more !  
    
  We will be discussing all these techniques later in one of the part of this series, so stay tuned …

#### Vectorization

As we discussed earlier that in order to make our machine learning algorithm make sense of text data, we need to convert the characters/words/sentences into numbers/vectors.  
  
So what exactly is **Vectorization** ?  
  
Vectorization is the process of conversion of words into numbers/vectors.

sample\_text = 'My name is Yash'

vectorized\_text = [123, 32, 15, 107]

Types of Vectorization methods :-

* Count Vectorization
* N-Gram Vectorization
* Bag Of Words
* TF-IDF Vectorization
* Word2Vec
* BERT (It is basically Word2Vec with Context) (state-of-the-art, Advance topic)

Again, we will be discussing all these techniques later in one of the part of this series, so stay tuned …

#### Machine Learning Algorithm

After we have successfully converted our raw text data into vectors, we are now ready to feed this vectorized data and it’s corresponding label like spam or not spam into our machine learning algorithm.  
  
There are many Machine Learning algorithms out there like Decision Trees, Random Forest Classifier, KNN, Logistic Regression etc, but the algorithms which works better on textual data are **Naive Bayes Algorithm** which works on the concept of Conditional Probability, **RNNs** (Recurrent Neural Networks), **LSTMs** (Long Short Term Memory Neural Networks), **LSTMs with Attention Mechanism**, **Transformers**, **BERT** etc.

After training with any of the mentioned Machine Learning/Deep Learning algorithms, We can now say that we have successfully trained our Spam classifier model which can detect whether a message/email is spam or not !

This is our NLP Pipeline.  
So this is all for this post folks !If you liked my content please do share and support.We will continue more stuff on NLP in my next part, until then, stay tuned :D.Happy learning !

Full code can be found [here](https://github.com/yash-007/NLP-with-Deep-Learning/blob/master/PREPROCESSING%20TECHNIQUES/All_text_cleaning_techniques.ipynb).

[Click here](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-3/) for part – 3 of this series.

In this part, we will be discussing about various Text Cleaning methods in Natural Processing Language with Python.  
  
Do checkout my [*previous blog*](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-2/) on this series where I have discussed in detail about the NLP Pipeline.

In this part I will be using the [SMSSpamCollection](https://github.com/yash-007/NLP-with-Deep-Learning/blob/master/PREPROCESSING%20TECHNIQUES/SMSSpamCollection) dataset and will be applying the following Text Pre-Processing/Cleaning techniques :-

* To Lower conversion
* Removal of HTML tags
* Removal of URLs
* Removal of Numbers
* Converting Numbers to Words
* Converting Accented characters to ASCII
* Expanding Contractions
* Stemming
* Lemmatizing
* Emoji Removal
* Removal of Punctuation
* Removing Stop words
* Removing Extra White Spaces

#### Lower case conversion

# importing necessary libraries

import os

import numpy as np

import pandas as pd

import nltk

nltk.download('punkt')

# importing/reading raw text data

data = pd.read\_csv("/content/SMSSpamCollection", sep="\t", header=None)

data.columns = ["category", "text"]

data.head()

category text

0 ham Go until jurong point, crazy.. Available

1 ham Ok lar... Joking wif u oni...

2 spam Free entry in 2 a wkly comp to win FA Cup

3 ham U dun say so early hor... U c already

4 ham Nah I don't think he goes to usf, he

def convert\_to\_lower(text):

return text.lower()

sample\_text = 'MY NAME IS YASH'

lowered = convert\_to\_lower(sample\_text)

print(lowered)

output : my name is yash

#### Removal of HTML tags

import re

punc = list(string.punctuation)

def remove\_html\_tags(text):

html\_pattern = r'<.\*?>'

without\_html = re.sub(pattern=html\_pattern, repl=' ', string=text)

return without\_html

sample\_text = 'Do you know that <my name> is <yash>'

print(remove\_html\_tags(sample\_text))

output : Do you know that is

#### Removal of URLs

import re

def remove\_urls(text):

url\_pattern = r'https?://\S+|www\.\S+'

without\_urls = re.sub(pattern=url\_pattern, repl=' ', string=text)

return without\_urls

sample\_text = 'do visit my webiste : www.webiste.com'

print(remove\_urls(sample\_text))

output : do visit my website :

#### Removal of Numbers

import re

def remove\_numbers(text):

number\_pattern = r'\d+'

without\_number = re.sub(pattern=number\_pattern, repl=" ", string=text)

return without\_number

sample\_text = 'My number is 98274610010 and pincode is 230012'

print(remove\_numbers(sample\_text))

output : My number is and pincode is

#### Converting numbers to words

# !pip install num2words

from num2words import num2words

def convert\_num\_2\_words(text):

splittedText = text.split()

for i in range(len(splittedText)):

if splittedText[i].isdigit():

splittedText[i] = num2words(splittedText[i])

num\_2\_words = ' '.join(splittedText)

return num\_2\_words

sample\_text = 'My lucky number is 7'

print(convert\_num\_2\_words(sample\_text))

output : My lucky number is seven

#### Converting accented characters to ASCII characters

# !pip install unidecode

import unidecode

def convert\_accented\_2\_ascii(text):

return unidecode.unidecode(text)

example\_text = "This is an example text with accented characters like dèèp lèarning ánd cömputer vísíön etc"

print(f"Original sentence: {example\_text}")

print(f"Converted sentence: {convert\_accented\_2\_ascii(example\_text)}")

output :

Original sentence: This is an example text with accented characters like dèèp lèarning ánd cömputer vísíön etc  
Converted sentence: This is an example text with accented characters like deep learning and computer vision etc

#### Expanding contractions

# !pip install contractions

import contractions

def expand\_contractions(text):

expanded\_text = []

for word in text.split():

expanded\_text.append(contractions.fix(word))

return " ".join(expanded\_text)

example\_text = "Sometimes our mind doesn't work properly. I've tried everything."

print(f"Original text: {example\_text}")

print(f"Expanded text: {expand\_contractions(example\_text)}")

output :

Original text: Sometimes our mind doesn’t work properly. I’ve tried everything.  
Expanded text: Sometimes our mind does not work properly. I have tried everything.

#### Stemming

**Stemming** is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma.

from nltk.stem import PorterStemmer

from nltk import word\_tokenize

def stemming(text):

stemmer = PorterStemmer()

tokens = word\_tokenize(text)

for i in range(len(tokens)):

stem\_word = stemmer.stem(tokens[i])

tokens[i] = stem\_word

return " ".join(tokens)

sample\_text = 'i love gaming and I recently visited my friend's place'

print(stemming(sample\_text))

output : i love gam and I recent visite my friend place

#### Lemmatizing

**Lemmatization** takes into consideration the morphological analysis of the words. To do so, it is necessary to have detailed dictionaries which the algorithm can look through to link the form back to its lemma.

Basic difference between stemming and lemmatizing is that in **stemming**, the removal of suffix takes place without any meaning. On the other hand, **lemmatizing** takes morphological and lexical meaning into consideration and then returns a much more meaning full ‘lemma’.

from nltk.stem import WordNetLemmatizer

from nltk import word\_tokenize

nltk.download("wordnet")

def lemmatizing(text):

lemmatizer = WordNetLemmatizer()

tokens = word\_tokenize(text)

for i in range(len(tokens)):

lemma\_word = lemmatizer.lemmatize(tokens[i])

tokens[i] = lemma\_word

return " ".join(tokens)

sample\_text = 'i love gaming and I recently visited my friend's place'

print(lemmatizing(sample\_text))

output : i love game and I recent visit my friend place

#### Emoji removal

import re

def remove\_emoji(text):

emoji\_pattern = re.compile("["

u"\U0001F600-\U0001F64F" # emoticons

u"\U0001F300-\U0001F5FF" # symbols & pictographs

u"\U0001F680-\U0001F6FF" # transport & map symbols

u"\U0001F1E0-\U0001F1FF" # flags (iOS)

u"\U00002500-\U00002BEF" # chinese char

u"\U00002702-\U000027B0"

u"\U00002702-\U000027B0"

u"\U000024C2-\U0001F251"

u"\U0001f926-\U0001f937"

u"\U00010000-\U0010ffff"

u"\u2640-\u2642"

u"\u2600-\u2B55"

u"\u200d"

u"\u23cf"

u"\u23e9"

u"\u231a"

u"\ufe0f" # dingbats

u"\u3030"

"]+", flags=re.UNICODE)

removeEmoji = emoji\_pattern.sub(r'', text)

return removeEmoji

example\_text = "This is a test 😻 "

print(f"Original text: {example\_text}")

print(f"Removed emoji: {remove\_emoji(example\_text)}")

output :  
Original text: This is a test 😻  
Removed emoji: This is a test

#### Punctuation Removal

import string

def remove\_punctuation(text):

return text.translate(str.maketrans('', '', string.punctuation))

example\_text = 'i love gaming, programming, and what more ? .'

print(remove\_punctuation(example\_text))

output : i love gaming programming and what more

#### Stop words removal

The words which are generally filtered out before processing a natural language are called **stop words**. These are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc) and does not add much information to the text. Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”.  
  
**Why do we need to remove stop-words ?**

Stop words are available in abundance in any human language. By removing these words, we remove the low-level information from our text in order to give more focus to the important information. In order words, we can say that the removal of such words does not show any negative consequences on the model we train for our task.

Removal of stop words definitely reduces the dataset size and thus reduces the training time due to the fewer number of tokens involved in the training.  
  
We do not always remove the stop words. The removal of stop words is highly dependent on the task we are performing and the goal we want to achieve.

from nltk.corpus import stopwords

from nltk import word\_tokenize

nltk.download("stopwords")

def remove\_stopwords(text):

removed = []

stop\_words = list(stopwords.words("english"))

tokens = word\_tokenize(text)

for i in range(len(tokens)):

if tokens[i] not in stop\_words:

removed.append(tokens[i])

return " ".join(removed)

example\_text = 'The movie was not good at all.'

print(remove\_stopwords(example\_text))

output : movie good

#### Removal of extra white space

def remove\_extra\_white\_spaces(text):

single\_char\_pattern = r'\s+[a-zA-Z]\s+'

without\_sc = re.sub(pattern=single\_char\_pattern, repl=" ", string=text)

return without\_sc

example\_text = 'i love food it makes me happy'

print(remove\_extra\_white\_spaces(example\_text))

output : i love food it makes me happy

So these were some of the most important text cleaning/pre-processing techniques which are applied on raw text data. That’s all for this post and we will soon see an **end to end implementation of spam classifier** using all these techniques.  
  
So stay tuned and support me !See you all in my next post !Happy Learning 😀

Full code can be found [here](https://github.com/yash-007/NLP-with-Deep-Learning/blob/master/PREPROCESSING%20TECHNIQUES/All_text_cleaning_techniques.ipynb).

[Click here](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-4/) for part – 4 of this series.

In my [last post](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-3/) we talked about various Text Cleaning techniques in which we discussed -> lower case conversion, removal of URLs, removal of HTML tags, removal of numbers, removal of emojis, punctuation, extra white spaces and so many more from the raw text.

In this post, we will discuss about various important Vectorization Techniques which are used to convert raw text to numbers in order to make sense for the computer.

#### What is Vectorization ?

**Vectorization** is the process of conversion of characters/words/sentences into vectors or numbers because our Machine Learning models cannot understand words, they need vectors to apply some mathematical equations and calculate derivatives in order to make sense of the text.

In this post, I’ll be exploring the following Vectorization Techniques :-

* Count Vectorization
* N-Gram Vectorization
* TF-IDF Vectorization
* Word2Vec Vectorization

I will be using the following example for all the techniques.

sample\_text = [

'My name is Yash Kelkar',

'I am in my final year of graduation',

'I like to play video games',

'My favourite outdoor sport is badminton'

]

#### Count Vectorization Technique :-

It is the simplest of all vectorization techniques.

* It creates a **Document Term Matrix**, now a doc-term matrix is a matrix whose **rows** are every single element our list, for example, we have **4** elements/docs in our ‘sample\_text’, **columns** are all the unique words from our whole document/whole ‘sample\_text’. Each cell of our Doc-Term matrix represents the **frequency** of that word in current cell.
* Now let’s code what we discussed above ! (It’s really simple!).

cv = CountVectorizer()

X = cv.fit\_transform(sample\_text)

X = X.toarray()

print(X.shape)

print(X)

output :-

(4, 20)

[[0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 0]

[1 0 0 1 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1]

[0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0]

[0 1 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 0]]

Let’s also print out the unique words :-

cols = cv.get\_feature\_names()

print(cols)

['am',

'badminton',

'favourite',

'final',

'games',

'graduation',

'in',

'is',

'kelkar',

'like',

'my',

'name',

'of',

'outdoor',

'play',

'sport',

'to',

'video',

'yash',

'year']

As you can see, we had 4 documents in our ‘sample\_text’ and the Count Vectorization technique has calculated 20 unique words from our whole documents, so therefore the shape : (4, 20). Now Each cell represents the frequency of that word, for example lets say for the cell row1 and col1 we have 0, this 0 represents that in our first doc i.e. ‘My name is Yash Kelkar’, ‘My’ word is not a unique word (we can verify this from above), so that is why it’s 0. Similarly for all the docs and words the process goes same. Hence we have our Count Vectorized matrix ready to be fed into our Machine Learning model.

#### N-gram Vectorization Technique :-

* It also creates a Document Term matrix.
* Columns represent all columns of adjacent words of length ‘n’.
* Rows represent each document in our sample\_text.
* Cell represent Count.

When n = 1, it is called **Uni-gram**, which is basically Count Vectorizer. Example = “my”, “name”, “is”, “yash”.   
When n = 2, it is called **Bi-gram**, Example = “my name”, “is yash”.  
When n = 3, it is called **Tri-gram**, Example = “my name is”, “yash”.  
(Now I think you got the idea :D)

So now let’s discuss the code.

ngram = CountVectorizer(ngram\_range=(1,3)) # (1,3) means we will consider all grams i.e. uni, bi and tri.

X = ngram.fit\_transform(sample\_text) # returns a sparse matrix, so we need to convert it to array.

X = X.toarray()

print(X.shape)

print(X)

output :-

(4, 54)

[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0]

[1 1 1 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 1 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1]

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0]

[0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1

1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0]]

As you can see, it creates a combination of 54 grams, so let’s take a look at these n-grams created.

ngrams = ngram.get\_feature\_names()

print(ngrams)

['am', 'am in', 'am in my', 'badminton', 'favourite', 'favourite outdoor', 'favourite outdoor sport', 'final', 'final year', 'final year of', 'games', 'graduation', 'in', 'in my', 'in my final', 'is', 'is badminton', 'is yash', 'is yash kelkar', 'kelkar', 'like', 'like to', 'like to play', 'my', 'my favourite', 'my favourite outdoor', 'my final', 'my final year', 'my name', 'my name is', 'name', 'name is', 'name is yash', 'of', 'of graduation', 'outdoor', 'outdoor sport', 'outdoor sport is', 'play', 'play video', 'play video games', 'sport', 'sport is', 'sport is badminton', 'to', 'to play', 'to play video', 'video', 'video games', 'yash', 'yash kelkar', 'year', 'year of', 'year of graduation']

Cool :D, so this was our n-grams and ‘X’ our n-gram doc-term matrix which is ready to be fed in our Machine Learning model.

#### TF – IDF Vectorization Technique :-

It stands for **Term Frequency – Inverse Document Frequency**. It is the most used and successful vectorization technique which can be applied to raw text data.

* It creates a document term matrix.
* Columns are individual unique words.
* Rows are the number of documents present in our small corpus.
* Cell contain/represent a ‘**weight**‘ which signifies how important a word is for an individual text message.  
  The formula for calculating this ‘**weight**‘ is given below :-

So, it basically calculates the importance for each word present in each document of our text corpus  
Now let’s discuss this approach in code :-

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()

X = tfidf.fit\_transform(sample\_text)

X = X.toarray()

print(X.shape)

print(X)

(4, 20)

[[0. 0. 0. 0. 0. 0.

0. 0.39278432 0.49819711 0. 0.31799276 0.49819711

0. 0. 0. 0. 0. 0.

0.49819711 0. ]

[0.39505606 0. 0. 0.39505606 0. 0.39505606

0.39505606 0. 0. 0. 0.25215917 0.

0.39505606 0. 0. 0. 0. 0.

0. 0.39505606]

[0. 0. 0. 0. 0.4472136 0.

0. 0. 0. 0.4472136 0. 0.

0. 0. 0.4472136 0. 0.4472136 0.4472136

0. 0. ]

[0. 0.44592216 0.44592216 0. 0. 0.

0. 0.35157015 0. 0. 0.28462634 0.

0. 0.44592216 0. 0.44592216 0. 0.

0. 0. ]]

Now, we know what is (4, 20) right ? I hope you know now :D.  
So this is our ‘X’ i.e. TF-IDF matrix which is ready to fed into… you know where :).

Let’s discuss our last vectorization technique for today which is …

#### Word2Vec Vectorization Technique :-

These are a set of neural network models that have the aim to represent words in the vector space. These models are highly efficient and performant in understanding the context and relation between words. Similar words are placed close together in the vector space while dissimilar words are placed wide apart.

It is so amazing to represent words that it is even able to identify key relationships such that:

King - Man + Woman = Queen

It is able to decipher that what a Man is to a King, a Woman is to a Queen. The respective relationships could be identified through these models.

There are two models in this class:

* **CBOW (Continuous Bag of Words):** The neural network takes a look at the surrounding words (say 2 to the left and 2 to the right) and predicts the word that comes in between.
* **Skip-grams:** The neural network takes in a word and then tries to predict the surrounding words.

The neural network has one input layer, 1 hidden layer and 1 output layer to train on the data and build the vectors.  
So let’s finally discuss it in code :-

I’ll be using the ‘gensim’ library to implement word2vec vectorization technique.  
It does not lowercase/tokenize the sentences, so I do the same. The tokenized sentences are then passed to the model. I’ve set the size of vector to be 2, window to be 3 which defines the distance upto which to look and sg = 0 uses the CBOW model.

from gensim.models import word2vec

for i, sentence in enumerate(sample\_text):

tokenized= []

for word in sentence.split(' '):

word = word.split('.')[0]

word = word.lower()

tokenized.append(word)

sample\_text[i] = tokenized

model = word2vec.Word2Vec(sample\_text, workers = 1, size = 2, min\_count = 1, window = 3, sg = 0)

similar\_word = model.wv.most\_similar('sport')[0]

print("Most common word to 'sport' is: {}".format(similar\_word[0]))

output :-

Most common word to 'sport' is: play

**So yeah, we have discussed 4 Vectorization Techniques today.**I hope it helps and please do support and share my posts if it is helpful.

Full code can be found [here](https://github.com/yash-007/NLP-with-Deep-Learning/blob/master/PREPROCESSING%20TECHNIQUES/Vectorization_techniques.ipynb).  
  
So this is al in this post, stay tuned for awesome stuff!Happy learning 😀

As we have discussed all the basics in our previous blogs ([PART-1](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-1/), [PART-2](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-2/), [PART-3](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-3/) and [PART-4](http://bugspeed.xyz/nlp-vectorization-techniques-part-4/)) which included importing of raw text data, NLP pipeline, text cleaning, text vectorization techniques and this is the last part of this series in which we will make our own **Spam Classifier from scratch using python** using all the techniques which we learned in my previous blogs.

So without any further delay, let’s dive in.

The first step will include import all the necessary libraries which we will be needing in this project.

import re

import string

import numpy as np

import pandas as pd

import nltk

nltk.download('punkt')

nltk.download('wordnet')

nltk.download('stopwords')

from nltk.stem import WordNetLemmatizer

from nltk import word\_tokenize

from nltk.corpus import stopwords

from nltk import word\_tokenize

Second step is to load our raw *spam classifier dataset* into our notebook or script.

df = pd.read\_csv("/content/SMSSpamCollection", sep="\t", header=None)

print(df.shape)

df.head()

output :

(5572, 2)

0 1

0 ham Go until jurong point, crazy.. Available only ...

1 ham Ok lar... Joking wif u oni...

2 spam Free entry in 2 a wkly comp to win FA Cup fina...

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 aro...

This is our raw text data which has 5572 rows/samples and 2 columns – 0 and 1, 0 is our ‘label’ and 1 is our ‘text or context’.

The next step is to clean this data in order to remove unnecessary/noisy data.

For this I’ll be taking help of my 3rd blog in which I explained about all the text cleaning methods. ([PART-3](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-3/))

1st method is – **converting to lower case.**

def convert\_to\_lower(text):

return text.lower()

2nd method – **removing numbers.**

def remove\_numbers(text):

number\_pattern = r'\d+'

without\_number = re.sub(pattern=number\_pattern, repl=" ", string=text)

return without\_number

3rd method – **lemmatizing.** (again I have explained all these text cleaning methods in [PART-3](http://bugspeed.xyz/natural-language-processing-beginner-to-advanced-part-3/))

def lemmatizing(text):

lemmatizer = WordNetLemmatizer()

tokens = word\_tokenize(text)

for i in range(len(tokens)):

lemma\_word = lemmatizer.lemmatize(tokens[i])

tokens[i] = lemma\_word

return " ".join(tokens)

4th method – **removing punctuation.**

def remove\_punctuation(text):

return text.translate(str.maketrans('', '', string.punctuation))

5th method – **removing stopwords.**

def remove\_stopwords(text):

removed = []

stop\_words = list(stopwords.words("english"))

tokens = word\_tokenize(text)

for i in range(len(tokens)):

if tokens[i] not in stop\_words:

removed.append(tokens[i])

return " ".join(removed)

6th and last method – **removing extra white spaces.**

def remove\_extra\_white\_spaces(text):

single\_char\_pattern = r'\s+[a-zA-Z]\s+'

without\_sc = re.sub(pattern=single\_char\_pattern, repl=" ", string=text)

return without\_sc

Now this is all the text cleaning methods I will be using for this dataset, ofcourse you can try and experiment any other methods, but to keep it short and simple, I have selected these 6 methods for now. Next step is to apply all these text cleaning methods to our dataset ‘df’.

df['context'] = df['context'].apply(lambda x: convert\_to\_lower(x))

df['context'] = df['context'].apply(lambda x: remove\_numbers(x))

df['context'] = df['context'].apply(lambda x: remove\_punctuation(x))

df['context'] = df['context'].apply(lambda x: remove\_stopwords(x))

df['context'] = df['context'].apply(lambda x: remove\_extra\_white\_spaces(x))

df['context'] = df['context'].apply(lambda x: lemmatizing(x))

Therefore, our raw noisy text data is no more noisy 😀

Now the next step is **Vectorization**. (you know by now what is vectorization right ? well, it is simply converting raw text/words to vectors/numbers so that our machine learning model can understand).  
I have already discussed the important vectorization techniques in my [PART-4](http://bugspeed.xyz/nlp-vectorization-techniques-part-4/).  
So let’s straight dive into code!

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer

In this project I will be using **n-gram vectorization** technique because it gave me around **93%** accuracy as compared to simple *count* *vectorization* and *tfidf vectorization*.

cv\_ngram = CountVectorizer(ngram\_range=(1,3))

X\_cv\_ngram = cv\_ngram.fit\_transform(df['context'])

X\_cv\_ngram = X\_cv\_ngram.toarray()

print(X\_cv\_ngram.shape)

output :

(5572, 68742)

Now, the vectorization is done, let’s **split the dataset** for our machine learning model.

X\_train\_ngram, X\_test\_ngram, y\_train\_ngram, y\_test\_ngram = train\_test\_split(X\_cv\_ngram, df['label'].values, test\_size=0.2)

FYI – I already converted the categorical labels to numerical, you can take a look at the full code provided in the end of this blog!

Now the interesting part – **Model Building.**  
I will be using **Naive Bayes algorithm** as it is considered to be good when dealing with text data.  
Again feel free to experiment with other machine learning algorithms like RandomForest, DecisionTrees etc.

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

naiveBayes = GaussianNB()

naiveBayes.fit(X\_train\_ngram, y\_train\_ngram)

y\_pred\_ngram = naiveBayes.predict(X\_test\_ngram)

Now we have our predicted values from our model, so let’s see the accuracy!

print(accuracy\_score(y\_test\_ngram, y\_pred\_ngram))

output :

0.9264573991031391

Great! It’s approximately 93% accurate.

Also keep in mind this was kind of an imbalanced dataset but in this blog we don’t care because I just wanted to show you how it is done – an end to end project with NLP and ofcourse we can deal with imbalanced dataset but first we needed to know how it is done all-together.

So this is all for this post and I hope you enjoyed it! If you did please do share and support so that It will motivate me to write more of these 😀

See you in the next blog, take care!

Full code can be found [here](https://github.com/yash-007/NLP-with-Deep-Learning/blob/master/PREPROCESSING%20TECHNIQUES/spam_classifier.ipynb).