Avinash Navlani
December 13th, 2019

PYTHON

Text Analytics for Beginners using NLTK

Python NTLK (natural language tooklit) sentiment analysis tutorial. Learn how to create and develop sentiment analysis using Python. Follow specific steps to mine and analyze text for natural language processing.

In today's area of internet and online services, data is generating at incredible speed and amount. Generally, Data analyst, engineer, and scientists are handling relational or tabular data. These tabular data columns have either numerical or categorical data. Generated data has a variety of structures such as text, image, audio, and video. Online activities such as articles, website text, blog posts, social media posts are generating unstructured textual data. Corporate and business need to analyze textual data to understand customer activities, opinion, and feedback to successfully derive their business. To compete with big textual data, text analytics is evolving at a faster rate than ever before.

Text Analytics has lots of applications in today's online world. By analyzing tweets on Twitter, we can find trending news and peoples reaction on a particular event. Amazon can understand user feedback or review on the specific product. BookMyShow can discover

In this tutorial, you are going to cover the following topics:

- Text Analytics and NLP
- Compare Text Analytics, NLP and Text Mining
  - Text Analysis Operations using NLTK
  - Tokenization
  - Stopwords
  - Lexicon Normalization such as Stemming and Lemmatization
  - POS Tagging
- Sentiment Analysis
- Text Classification
- Performing Sentiment Analysis using Text Classification

Text Analytics and NLP

Text communication is one of the most popular forms of day to day conversion. We chat, message, tweet, share status, email, write blogs, share opinion and feedback in our daily routine. All of these activities are generating text in a significant amount, which is unstructured in nature. I this area of the online marketplace and social media, It is essential to analyze vast quantities of data, to understand peoples opinion.

NLP enables the computer to interact with humans in a natural manner. It helps the computer to understand the human language and derive meaning from it. NLP is applicable in several problematic from speech recognition, language translation, classifying documents to information extraction. Analyzing movie review is one of the classic examples to demonstrate a simple NLP Bag-of-words model, on movie reviews.

Text mining also referred to as text analytics. Text mining is a process of exploring sizeable textual data and find patterns. Text Mining process the text itself, while NLP process with the underlying metadata. Finding frequency counts of words, length of the sentence, presence/absence of specific words is known as text mining. Natural language processing is one of the components of text mining. NLP helps identified sentiment, finding entities in the sentence, and category of blog/article. Text mining is preprocessed data for text analytics. In Text Analytics, statistical and machine learning algorithm used to classify information.

Text Analysis Operations using NLTK

NLTK is a powerful Python package that provides a set of diverse natural languages algorithms. It is free, opensource, easy to use, large community, and well documented. NLTK consists of the most common algorithms such as tokenizing, part-of-speech tagging, stemming, sentiment analysis, topic segmentation, and named entity recognition. NLTK helps the computer to analysis, preprocess, and understand the written text.

```
!pip install nltk
```

```
Requirement already satisfied: nltk in /home/northout/anaconda2/lib/python2.7/site-packages
Requirement already satisfied: six in /home/northout/anaconda2/lib/python2.7/site-packages (fr
 [33mYou are using pip version 9.0.1, however version 10.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command. [0m
```

```
#Loading NLTK
import nltk
```

Tokenization

Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks such as words or sentence is called Tokenization. Token is a single entity that is building blocks for sentence or paragraph.

Sentence tokenizer breaks text paragraph into sentences.

```
from nltk.tokenize import sent_tokenize
text="""Hello Mr. Smith, how are you doing today? The weather is great, and city is awesome.
The sky is pinkish-blue. You shouldn't eat cardboard"""
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

```
['Hello Mr. Smith, how are you doing today?', 'The weather is great, and city is awesome.', 'T
```

Here, the given text is tokenized into sentences.

Word Tokenization

Word tokenizer breaks text paragraph into words.

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

```
['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', '
```

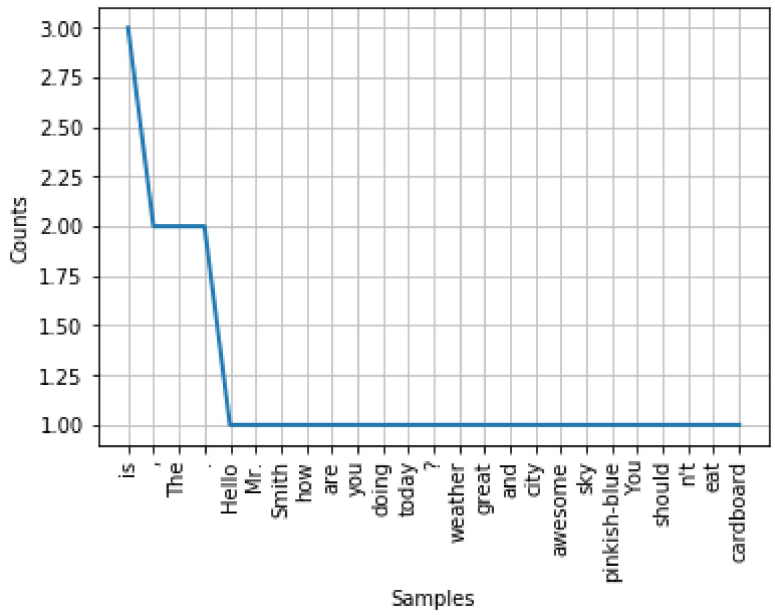
Frequency Distribution

```
from nltk.probability import FreqDist
fdist = FreqDist(tokenized_word)
print(fdist)
```

```
<FreqDist with 25 samples and 30 outcomes>
```

```
fdist.most_common(2)
```

```
# Frequency Distribution Plot
import matplotlib.pyplot as plt
fdist.plot(30,cumulative=False)
plt.show()
```



Stopwords

Stopwords considered as noise in the text. Text may contain stop words such as is, am, are, this, a, an, the, etc.

In NLTK for removing stopwords, you need to create a list of stopwords and filter out your list of tokens from these words.

```
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

```
{'their', 'then', 'not', 'ma', 'here', 'other', 'won', 'up', 'weren', 'being', 'we', 'those',
```

```
filtered_sent=[]
for w in tokenized_sent:
    if w not in stop_words:
        filtered_sent.append(w)
print("Tokenized Sentence:",tokenized_sent)
print("Filterd Sentence:",filtered_sent)
```

```
Tokenized Sentence: ['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?']
Filterd Sentence: ['Hello', 'Mr.', 'Smith', ',', 'today', '?']
```

Lexicon Normalization

Lexicon normalization considers another type of noise in the text. For example, connection, connected, connecting word reduce to a common word "connect". It reduces derivationally related forms of a word to a common root word.

Stemming

Stemming is a process of linguistic normalization, which reduces words to their word root word or chops off the derivational affixes. For example, connection, connected, connecting word reduce to a common word "connect".

```
# Stemming
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize

ps = PorterStemmer()

stemmed_words=[]
for w in filtered_sent:
    stemmed_words.append(ps.stem(w))

print("Filtered Sentence:",filtered_sent)
print("Stemmed Sentence:",stemmed_words)
```

```
Stemmed Sentence: ['hello', 'mr.', 'smith', ',', 'today', '?']
```

Lemmatization

Lemmatization reduces words to their base word, which is linguistically correct lemmas. It transforms root word with the use of vocabulary and morphological analysis. Lemmatization is usually more sophisticated than stemming. Stemmer works on an individual word without knowledge of the context. For example, The word "better" has "good" as its lemma. This thing will miss by stemming because it requires a dictionary look-up.

```
#Lexicon Normalization
#performing stemming and Lemmatization

from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()

from nltk.stem.porter import PorterStemmer
stem = PorterStemmer()

word = "flying"
print("Lemmatized Word:",lem.lemmatize(word,"v"))
print("Stemmed Word:",stem.stem(word))


Lemmatized Word: fly
Stemmed Word: fli
```

POS Tagging

The primary target of Part-of-Speech(POS) tagging is to identify the grammatical group of a given word. Whether it is a NOUN, PRONOUN, ADJECTIVE, VERB, ADVERBS, etc. based on the context. POS Tagging looks for relationships within the sentence and assigns a corresponding tag to the word.

```
tokens=nltk.word_tokenize(sent)
print(tokens)
```

```
['Albert', 'Einstein', 'was', 'born', 'in', 'Ulm', ',', 'Germany', 'in', '1879', '.']
```

```
nltk.pos_tag(tokens)
```

```
[('Albert', 'NNP'),
 ('Einstein', 'NNP'),
 ('was', 'VBD'),
 ('born', 'VBN'),
 ('in', 'IN'),
 ('Ulm', 'NNP'),
 (',', ','),
 ('Germany', 'NNP'),
 ('in', 'IN'),
 ('1879', 'CD'),
 ('.', '.')]
```

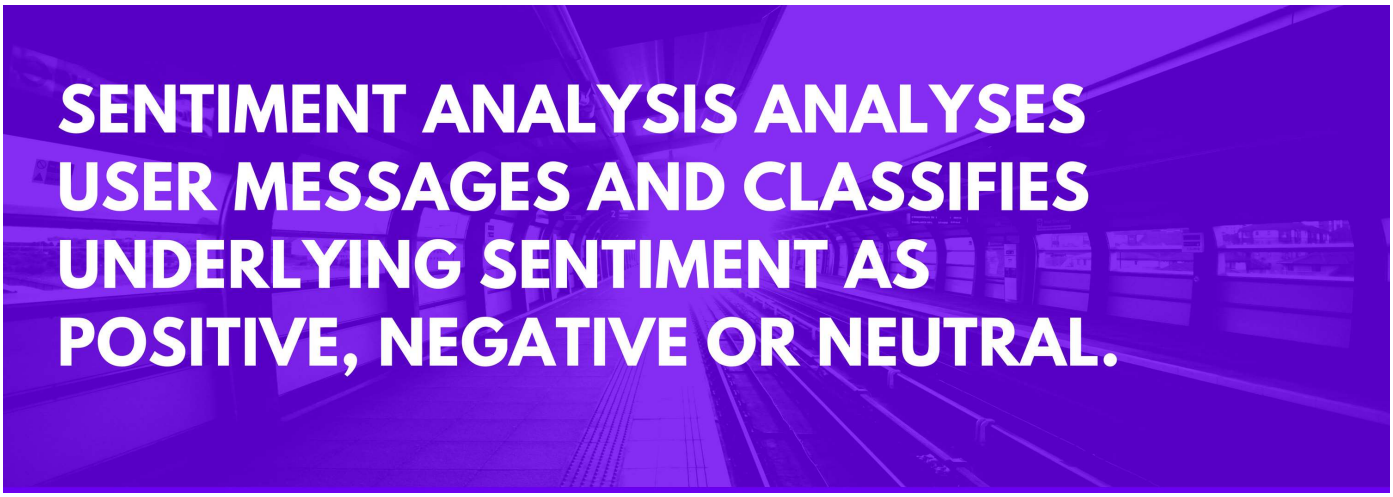
POS tagged: Albert/NNP Einstein/NNP was/VBD born/VBN in/IN Ulm/NNP ,/, Germany/NNP in/IN 1879/CD ./.

Sentiment Analysis

Nowadays companies want to understand, what went wrong with their latest products? What users and the general public think about the latest feature? You can quantify such information with reasonable accuracy using sentiment analysis.

Quantifying users content, idea, belief, and opinion is known as sentiment analysis. User's online post, blogs, tweets, feedback of product helps business people to the target audience and innovate in products and services. Sentiment analysis helps in understanding people in a

Human communication just not limited to words, it is more than words. Sentiments are combination words, tone, and writing style. As a data analyst, It is more important to understand our sentiments, what it really means?



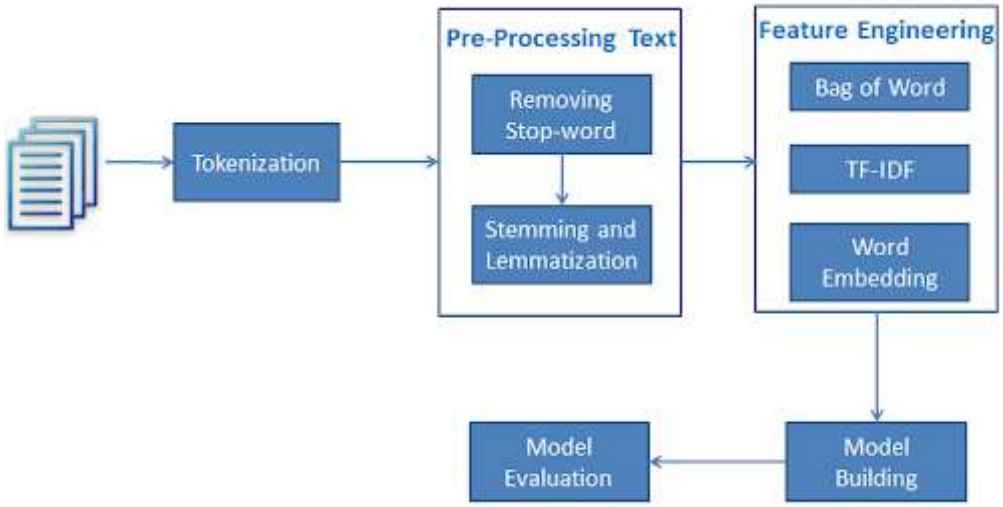
There are mainly two approaches for performing sentiment analysis.

- Lexicon-based: count number of positive and negative words in given text and the larger count will be the sentiment of text.
- Machine learning based approach: Develop a classification model, which is trained using the pre-labeled dataset of positive, negative, and neutral.

In this Tutorial, you will use the second approach(Machine learning based approach). This is how you learn sentiment and text classification with a single example.

Text Classification

Text classification is one of the important tasks of text mining. It is a supervised approach. Identifying category or class of given text such as a blog, book, web page, news articles, and tweets. It has various application in today's computer world such as spam detection, task categorization in CRM services, categorizing products on E-retailer websites, classifying the



Performing Sentiment Analysis using Text Classification

```
# Import pandas
import pandas as pd
```

Loading Data

Till now, you have learned data pre-processing using NLTK. Now, you will learn Text Classification. You will perform Multi-Nomial Naive Bayes Classification using scikit-learn.

In the model the building part, you can use the "Sentiment Analysis of Movie, Reviews" dataset available on Kaggle. The dataset is a tab-separated file. Dataset has four columns PhraseId, SentenceId, Phrase, and Sentiment.

This data has 5 sentiment labels:

0 - negative 1 - somewhat negative 2 - neutral 3 - somewhat positive 4 - positive

Here, you can build a model to classify the type of cultivar. The dataset is available on Kaggle. You can download it from the following link: https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data

```
data.head()
```

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

```
data.info()
```

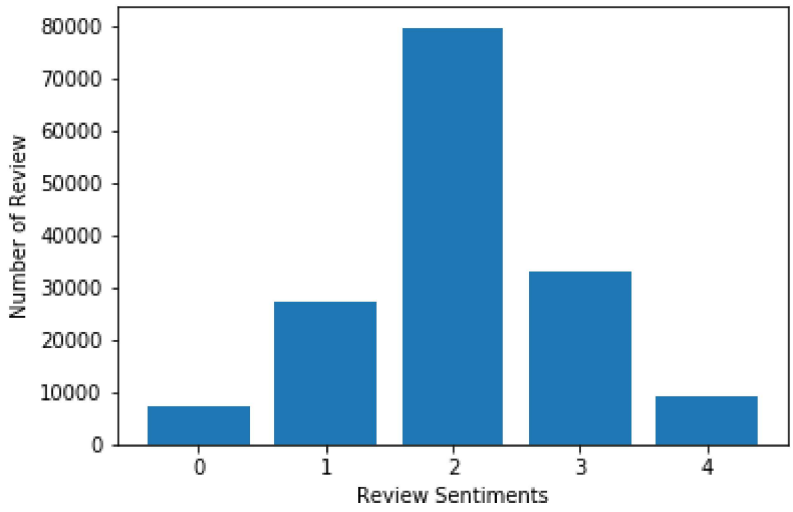
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 156060 entries, 0 to 156059
Data columns (total 4 columns):
PhraseId      156060 non-null int64
SentenceId    156060 non-null int64
Phrase        156060 non-null object
Sentiment     156060 non-null int64
dtypes: int64(3), object(1)
memory usage: 4.8+ MB
```

```
data.Sentiment.value_counts()
```

```
2    79582
3    32927
1    27273
```

```
Name: Sentiment, dtype: int64
```

```
Sentiment_count=data.groupby('Sentiment').count()
plt.bar(Sentiment_count.index.values, Sentiment_count['Phrase'])
plt.xlabel('Review Sentiments')
plt.ylabel('Number of Review')
plt.show()
```



Feature Generation using Bag of Words

In the Text Classification Problem, we have a set of texts and their respective labels. But we directly can't use text for our model. You need to convert these text into some numbers or vectors of numbers.

Bag-of-words model(BoW) is the simplest way of extracting features from the text. BoW converts text into the matrix of occurrence of words within a document. This model concerns about whether given words occurred or not in the document.

Example: There are three documents:

Doc 1: I love dogs. Doc 2: I hate dogs and knitting. Doc 3: Knitting is my hobby and passion.

	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	2	1	1

This matrix is using a single word. It can be a combination of two or more words, which is called a bigram or trigram model and the general approach is called the n-gram model.

You can generate document term matrix by using scikit-learn's CountVectorizer.

```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import RegexpTokenizer
#tokenizer to remove unwanted elements from out data like symbols and numbers
token = RegexpTokenizer(r'[a-zA-Z0-9]+')
cv = CountVectorizer(lowercase=True,stop_words='english',ngram_range = (1,1),tokenizer = token
text_counts= cv.fit_transform(data['Phrase'])
```

Split train and test set

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Let's split dataset by using function train_test_split(). You need to pass basically 3 parameters features, target, and test_set size. Additionally, you can use random_state to select records randomly.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    text_counts, data['Sentiment'], test_size=0.3, random_state=1)
```

Let's build the Text Classification Model using TF-IDF.

First, import the MultinomialNB module and create a Multinomial Naive Bayes classifier object using MultinomialNB() function.

Then, fit your model on a train set using fit() and perform prediction on the test set using predict().

```
from sklearn.naive_bayes import MultinomialNB
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Generation Using Multinomial Naive Bayes
clf = MultinomialNB().fit(X_train, y_train)
predicted= clf.predict(X_test)
print("MultinomialNB Accuracy:",metrics.accuracy_score(y_test, predicted))
```

```
MultinomialNB Accuracy: 0.604916912299
```

Well, you got a classification rate of 60.49% using CountVector(or BoW), which is not considered as good accuracy. We need to improve this.

Feature Generation using TF-IDF

In Term Frequency(TF), you just count the number of words occurred in each document. The main issue with this Term Frequency is that it will give more weight to longer documents. Term frequency is basically the output of the BoW model.

IDF(Inverse Document Frequency) measures the amount of information a given word provides across the document. IDF is the logarithmically scaled inverse ratio of the number of documents that contain the word and the total number of documents.

$$\text{idf(W)} = \log \frac{\#(\text{documents})}{\#(\text{documents containing word W})}$$

times occurred in given documents and must be absent in the other documents. So the words must be a signature word.

	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	0.18	**0.48**	0.18							
Doc 2	0.18		0.18	**0.48**	0.18	0.18				
Doc 3					0.18	0.18	**0.48**	**0.95**	**0.48**	**0.48**

```
from sklearn.feature_extraction.text import TfidfVectorizer
tf=TfidfVectorizer()
text_tf= tf.fit_transform(data['Phrase'])
```

Split train and test set (TF-IDF)

Let's split dataset by using function train_test_split(). You need to pass basically 3 parameters features, target, and test_set size. Additionally, you can use random_state to select records randomly.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    text_tf, data['Sentiment'], test_size=0.3, random_state=123)
```

Model Building and Evaluation (TF-IDF)

Let's build the Text Classification Model using TF-IDF.

First, import the MultinomialNB module and create the Multinomial Naive Bayes classifier object using MultinomialNB() function.

Then, fit your model on a train set using fit() and perform prediction on the test set using predict().

```
from sklearn import metrics
# Model Generation Using Multinomial Naive Bayes
clf = MultinomialNB().fit(X_train, y_train)
predicted= clf.predict(X_test)
print("MultinomialNB Accuracy:",metrics.accuracy_score(y_test, predicted))


MultinomialNB Accuracy: 0.586526549618
```

Well, you got a classification rate of 58.65% using TF-IDF features, which is not considered as good accuracy. We need to improve the accuracy by using some other preprocessing or feature engineering. Let's suggest in comment box some approach for accuracy improvement.

Conclusion

Congratulations, you have made it to the end of this tutorial!

In this tutorial, you have learned what Text Analytics is, NLP and text mining, basics of text analytics operations using NLTK such as Tokenization, Normalization, Stemming, Lemmatization and POS tagging. What are sentiment analysis and text classification using scikit-learn?

I look forward to hearing any feedback or questions. You can ask the question by leaving a comment, and I will try my best to answer it.

If you are interested in learning more about Python, please take DataCamp's Natural Language Processing Fundamentals in Python course.

Use DataCamp Workspace to experiment with the code in this tutorial!

Open in Workspace

Subscribe to RSS

About Terms Privacy