



4222 – SURYA GROUP OF INSTITUTIONS

VIKRAVANDI

## SENTIMENT ANALYSIS FOR MARKETING

Phase 4: Development part 2

Prepared by,

S.DHANUSH

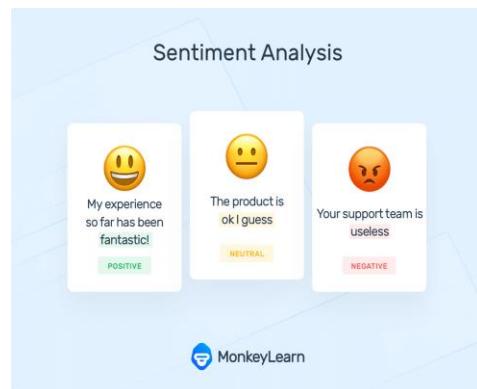
REG. NO: 42221106004

ECE 3<sup>rd</sup> year

## AI Phase 4: Development part 2

What is NLP for sentiment analysis:

Sentiment analysis (or opinion mining) is a natural language processing (NLP) 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.



What is generating insights in sentiment analysis:

Sentiment analysis solutions apply consistent criteria to generate more accurate insights. For example, a machine learning model can be trained to recognise that there are two aspects with two different sentiments. It would average the overall sentiment as neutral, but also keep track of the details.

Which algorithm is used for sentiment analysis in NLP

Overall, Sentiment analysis may involve the following types of classification algorithms: Linear Regression. Naive Bayes. Support Vector Machines.

How to use NLP to do sentiment analysis?



In sentiment analysis, Natural Language Processing (NLP) is essential. NLP uses computational methods to interpret and comprehend human language. It includes several operations, including sentiment analysis, named entity recognition, part-of-speech tagging, and tokenization.

NLP methods for sentiment analysis

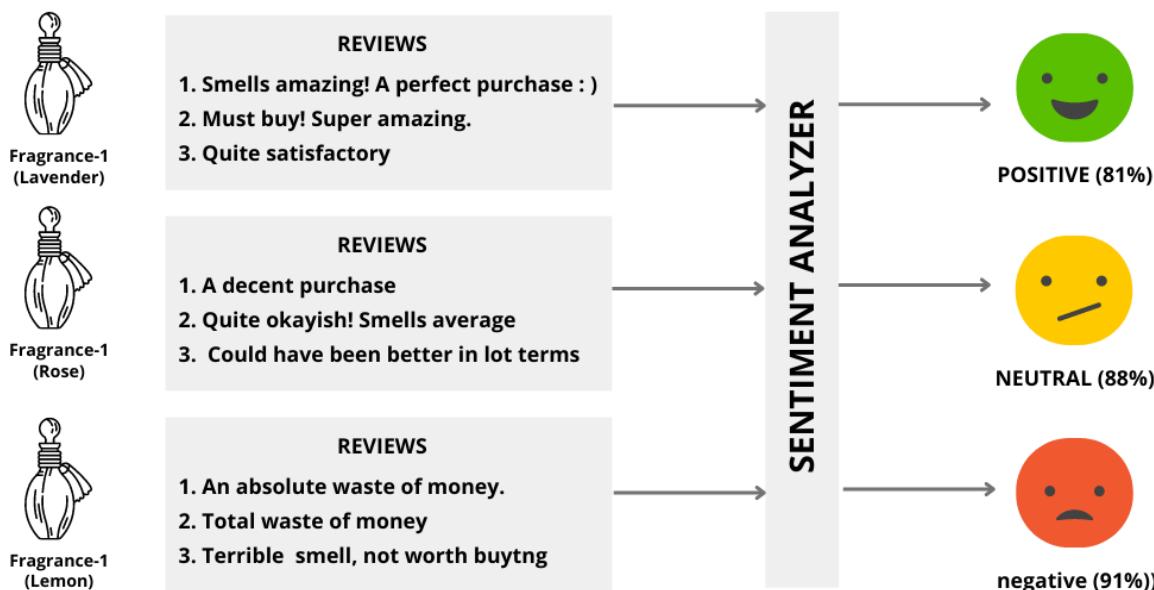
1. using a dictionary of manually defined keywords,
2. creating a 'bag of words',
3. using the TF-IDF strategy.

## Gaining Insights and Making Decisions with Sentiment Analysis

Well, by now I guess we are somewhat accustomed to what sentiment analysis is. But what is its significance and how do organizations benefit from it? Let us try and explore the same with an example. Assume you start a company that sells perfumes on an online platform. You put up a wide range of fragrances out there and soon customers start flooding in. After some time you decide to change the pricing strategy of perfumes—you plan to increase the prices of the popular fragrances and at the same time offer discounts on unpopular ones. Now, in order to determine which fragrances are popular, you start going through customer reviews of all the

fragrances. But you're stuck! They are just so many that you cannot go through them all in one lifetime. This is where sentiment analysis can rope you out of the pit.

You simply gather all the reviews in one place and apply sentiment analysis to it. The following is a schematic representation of sentiment analysis on the reviews of three fragrances of perfumes—Lavender, Rose, and Lemon. (Please note that these reviews might have incorrect spellings, grammar, and punctuations as it is in the real-world scenarios)



From these results, we can clearly see that:

- Fragrance-1 (Lavender) has highly positive reviews by the customers which indicates your company can escalate its prices given its popularity.
- Fragrance-2 (Rose) happens to have a neutral outlook amongst the customer which means your company should not change its pricing.
- Fragrance-3 (Lemon) has an overall negative sentiment associated with it—thus, your company should consider offering a discount on it to balance the scales.

This was just a simple example of how sentiment analysis can help you gain insights into your products/services and help your organization make decisions.

Why is natural language processing important?

Businesses use massive quantities of unstructured, text-heavy data and need a way to efficiently process it. A lot of the information created online and stored in databases is natural human language, and until recently, businesses could not effectively analyze this data. This is where natural language processing is useful.

## Features

The advantage of natural language processing can be seen when considering the following two statements: "Cloud computing insurance should be part of every service-level agreement," and, "A good SLA ensures an easier night's sleep -- even in the cloud." If a user relies on natural language processing for search, the program will recognize that *cloud computing* is an entity, that *cloud* is an abbreviated form of cloud computing and that *SLA* is an industry acronym for service-level agreement.

### Step 1 — Installing NLTK and Downloading the Data

You will use the NLTK package in Python for all NLP tasks in this tutorial. In this step you will install NLTK and download the sample tweets that you will use to train and test your model.

First, install the NLTK package with the pip package manager:

```
1. pip install nltk==3.3
```

This tutorial will use sample tweets that are part of the NLTK package. First, start a Python interactive session by running the following command:

```
1. python3  
2.
```

Then, import the `nltk` module in the python interpreter.

## 1. import nltk

Download the sample tweets from the NLTK package:

```
1. nltk.download('twitter_samples')
```

Running this command from the Python interpreter downloads and stores the tweets locally. Once the samples are downloaded, they are available for your use.

You will use the negative and positive tweets to train your model on sentiment analysis later in the tutorial. The tweets with no sentiments will be used to test your model.

If you would like to use your own dataset, you can gather tweets from a specific time period, user, or hashtag by using the Twitter API.

Now that you've imported NLTK and downloaded the sample tweets, exit the interactive session by entering in `exit()`. You are ready to import the tweets and begin processing the data.

### Step 2 — Tokenizing the Data

Language in its original form cannot be accurately processed by a machine, so you need to process the language to make it easier for the machine to understand. The first part of making sense of the data is through a process called *tokenization*, or splitting strings into smaller parts called *tokens*.

A token is a sequence of characters in text that serves as a unit. Based on how you create the tokens, they may consist of words, emoticons, hashtags, links, or even individual characters. A basic way of breaking language into tokens is by splitting the text based on whitespace and punctuation.

To get started, create a new `.py` file to hold your script. This tutorial will use `nlp_test.py`:

1. `nano nlp_test.py`
2. In this file, you will first import the `twitter_samples` so you can work with that data:

```
nlp_test.py
```

```
from nltk.corpus import twitter_samples
```

Copy

This will import three datasets from NLTK that contain various tweets to train and test the model:

- negative\_tweets.json: 5000 tweets with negative sentiments
- positive\_tweets.json: 5000 tweets with positive sentiments
- tweets.20150430-223406.json: 20000 tweets with no sentiments

Next, create variables for positive\_tweets, negative\_tweets, and text:

nlp\_test.py

```
from nltk.corpus import twitter_samples
```

```
positive_tweets = twitter_samples.strings('positive_tweets.json')
negative_tweets = twitter_samples.strings('negative_tweets.json')
text = twitter_samples.strings('tweets.20150430-223406.json')
```

The strings() method of twitter\_samples will print all of the tweets within a dataset as strings. Setting the different tweet collections as a variable will make processing and testing easier.

Before using a tokenizer in NLTK, you need to download an additional resource, punkt. The punkt module is a pre-trained model that helps you tokenize words and sentences. For instance, this model knows that a name may contain a period (like “S. Daityari”) and the presence of this period in a sentence does not necessarily end it. First, start a Python interactive session:

1. python3
- 2.

Copy

Run the following commands in the session to download the punkt resource:

1. import nltk
- 2.
3. nltk.download('punkt')
- 4.

Copy

Once the download is complete, you are ready to use NLTK’s tokenizers. NLTK provides a default tokenizer for tweets with the .tokenized() method. Add a line to create an object that tokenizes the positive\_tweets.json dataset:

### nlp\_test.py

```
from nltk.corpus import twitter_samples

positive_tweets = twitter_samples.strings('positive_tweets.json')
negative_tweets = twitter_samples.strings('negative_tweets.json')
text = twitter_samples.strings('tweets.20150430-223406.json')
tweet_tokens = twitter_samples.tokenized('positive_tweets.json')
```

If you'd like to test the script to see the `.tokenized` method in action, add the highlighted content to your `nlp_test.py` script. This will tokenize a single tweet from the `positive_tweets.json` dataset:

### nlp\_test.py

```
from nltk.corpus import twitter_samples

positive_tweets = twitter_samples.strings('positive_tweets.json')
negative_tweets = twitter_samples.strings('negative_tweets.json')
text = twitter_samples.strings('tweets.20150430-223406.json')
tweet_tokens = twitter_samples.tokenized('positive_tweets.json')[0]

print(tweet_tokens[0])
```

Save and close the file, and run the script:

1. `python3 nlp_test.py`
- 2.

Copy

The process of tokenization takes some time because it's not a simple split on white space. After a few moments of processing, you'll see the following:

### Output

```
['#FollowFriday',
 '@France_Inte',
 '@PKuchly57',
 '@Milipol_Paris',
 'for',
 'being',
 'top',
```

```
'engaged',
'members',
'in',
'my',
'community',
'this',
'week',
':)']
```

Here, the `.tokenized()` method returns special characters such as `@` and `_`. These characters will be removed through regular expressions later in this tutorial.

Now that you've seen how the `.tokenized()` method works, make sure to comment out or remove the last line to print the tokenized tweet from the script by adding a `#` to the start of the line:

```
nlp_test.py
from nltk.corpus import twitter_samples

positive_tweets = twitter_samples.strings('positive_tweets.json')
negative_tweets = twitter_samples.strings('negative_tweets.json')
text = twitter_samples.strings('tweets.20150430-223406.json')
tweet_tokens = twitter_samples.tokenized('positive_tweets.json')[0]

#print(tweet_tokens[0])
```

Your script is now configured to tokenize data. In the next step you will update the script to normalize the data.

### Step 3 — Normalizing the Data

Words have different forms—for instance, “ran”, “runs”, and “running” are various forms of the same verb, “run”. Depending on the requirement of your analysis, all of these versions may need to be converted to the same form,

“run”. *Normalization* in NLP is the process of converting a word to its canonical form.

Normalization helps group together words with the same meaning but different forms. Without normalization, “ran”, “runs”, and “running” would be treated as different words, even though you may want them to be treated as the same word. In

this section, you explore *stemming* and *lemmatization*, which are two popular techniques of normalization.

Stemming is a process of removing affixes from a word. Stemming, working with only simple verb forms, is a heuristic process that removes the ends of words.

In this tutorial you will use the process of lemmatization, which normalizes a word with the context of vocabulary and morphological analysis of words in text. The lemmatization algorithm analyzes the structure of the word and its context to convert it to a normalized form. Therefore, it comes at a cost of speed. A comparison of stemming and lemmatization ultimately comes down to a trade off between speed and accuracy.

Before you proceed to use lemmatization, download the necessary resources by entering the following in to a Python interactive session:

```
1. python3  
2.
```

Copy

Run the following commands in the session to download the resources:

```
1. import nltk  
2.  
3. nltk.download('wordnet')  
4.  
5. nltk.download('averaged_perceptron_tagger')  
6.
```

Copy

`wordnet` is a lexical database for the English language that helps the script determine the base word. You need the `averaged_perceptron_tagger` resource to determine the context of a word in a sentence.

Once downloaded, you are almost ready to use the lemmatizer. Before running a lemmatizer, you need to determine the context for each word in your text. This is achieved by a tagging algorithm, which assesses the relative position of a word in a sentence. In a Python session, Import the `pos_tag` function, and provide a list of tokens as an argument to get the tags. Let us try this out in Python:

```
1. from nltk.tag import pos_tag
2.
3. from nltk.corpus import twitter_samples
4.
5.
6.
7. tweet_tokens = twitter_samples.tokenized('positive_tweets.json')
8.
9. print(pos_tag(tweet_tokens[0]))
10.
```

Copy

Here is the output of the `pos_tag` function.

Output

```
[('#FollowFriday', 'JJ'),
('@France_Inte', 'NNP'),
('@PKuchly57', 'NNP'),
('@Milipol_Paris', 'NNP'),
('for', 'IN'),
('being', 'VBG'),
('top', 'JJ'),
('engaged', 'VBN'),
('members', 'NNS'),
('in', 'IN'),
('my', 'PRP$'),
('community', 'NN'),
('this', 'DT'),
('week', 'NN'),
(:)', 'NN')]
```

From the list of tags, here is the list of the most common items and their meaning:

- NNP: Noun, proper, singular
- NN: Noun, common, singular or mass
- IN: Preposition or conjunction, subordinating
- VBG: Verb, gerund or present participle
- VBN: Verb, past participle

Here is a full list of the dataset.

In general, if a tag starts with NN, the word is a noun and if it starts with VB, the word is a verb. After reviewing the tags, exit the Python session by entering exit().

To incorporate this into a function that normalizes a sentence, you should first generate the tags for each token in the text, and then lemmatize each word using the tag.

Update the nlp\_test.py file with the following function that lemmatizes a sentence:

nlp\_test.py

...

```
from nltk.tag import pos_tag
from nltk.stem.wordnet import WordNetLemmatizer

def lemmatize_sentence(tokens):
    lemmatizer = WordNetLemmatizer()
    lemmatized_sentence = []
    for word, tag in pos_tag(tokens):
        if tag.startswith('NN'):
            pos = 'n'
        elif tag.startswith('VB'):
            pos = 'v'
        else:
            pos = 'a'
        lemmatized_sentence.append(lemmatizer.lemmatize(word, pos))
    return lemmatized_sentence

print(lemmatize_sentence(tweet_tokens[0]))
```

Copy

This code imports the WordNetLemmatizer class and initializes it to a variable, lemmatizer.

The function lemmatize\_sentence first gets the position tag of each token of a tweet. Within the if statement, if the tag starts with NN, the token is assigned as a noun. Similarly, if the tag starts with VB, the token is assigned as a verb.

Save and close the file, and run the script:

```
1. python3 nlp_test.py  
2.
```

Copy

Here is the output:

```
Output  
['#FollowFriday',  
 '@France_Inte',  
 '@PKuchly57',  
 '@Milipol_Paris',  
 'for',  
 'be',  
 'top',  
 'engage',  
 'member',  
 'in',  
 'my',  
 'community',  
 'this',  
 'week',  
 ':)']
```

You will notice that the verb `being` changes to its root form, `be`, and the noun `members` changes to `member`. Before you proceed, comment out the last line that prints the sample tweet from the script.

Now that you have successfully created a function to normalize words, you are ready to move on to remove noise.

#### Step 4 — Removing Noise from the Data

In this step, you will remove noise from the dataset. *Noise* is any part of the text that does not add meaning or information to data.

Noise is specific to each project, so what constitutes noise in one project may not be in a different project. For instance, the most common words in a language are called *stop words*. Some examples of stop words are “is”, “the”, and “a”. They are

generally irrelevant when processing language, unless a specific use case warrants their inclusion.

In this tutorial, you will use regular expressions in Python to search for and remove these items:

- Hyperlinks - All hyperlinks in Twitter are converted to the URL shortener t.co. Therefore, keeping them in the text processing would not add any value to the analysis.
- Twitter handles in replies - These Twitter usernames are preceded by a @ symbol, which does not convey any meaning.
- Punctuation and special characters - While these often provide context to textual data, this context is often difficult to process. For simplicity, you will remove all punctuation and special characters from tweets.

To remove hyperlinks, you need to first search for a substring that matches a URL starting with http:// or https://, followed by letters, numbers, or special characters. Once a pattern is matched, the .sub() method replaces it with an empty string. Since we will normalize word forms within the remove\_noise() function, you can comment out the lemmatize\_sentence() function from the script.

Add the following code to your nlp\_test.py file to remove noise from the dataset:

```
nlp_test.py  
...  
  
import re, string  
  
def remove_noise(tweet_tokens, stop_words =()):  
  
    cleaned_tokens = []  
  
    for token, tag in pos_tag(tweet_tokens):  
        token = re.sub('http[s]?://(?:[a-zA-Z][0-9]|[$-_@.&+#!][!*\\(),]|"\'(?:%[0-9a-fA-F][0-9a-fA-F])+", token)  
        token = re.sub("(@[A-Za-z0-9_]+)", "", token)  
  
        if tag.startswith("NN"):  
            pos = 'n'  
        elif tag.startswith('VB'):  
            pos = 'v'
```

```
    pos = 'v'  
else:  
    pos = 'a'  
  
lemmatizer = WordNetLemmatizer()  
token = lemmatizer.lemmatize(token, pos)  
  
if len(token) > 0 and token not in string.punctuation and token.lower() not in  
stop_words:  
    cleaned_tokens.append(token.lower())  
return cleaned_tokens
```

Copy

This code creates a `remove_noise()` function that removes noise and incorporates the normalization and lemmatization mentioned in the previous section. The code takes two arguments: the tweet tokens and the tuple of stop words.

The code then uses a loop to remove the noise from the dataset. To remove hyperlinks, the code first searches for a substring that matches a URL starting with `http://` or `https://`, followed by letters, numbers, or special characters. Once a pattern is matched, the `.sub()` method replaces it with an empty string, or `"`.

Similarly, to remove `@` mentions, the code substitutes the relevant part of text using regular expressions. The code uses the `re` library to search `@` symbols, followed by numbers, letters, or `_`, and replaces them with an empty string. Finally, you can remove punctuation using the library `string`.

In addition to this, you will also remove stop words using a built-in set of stop words in NLTK, which needs to be downloaded separately.

Execute the following command from a Python interactive session to download this resource:

```
1. nltk.download('stopwords')  
2.
```

Copy

Once the resource is downloaded, exit the interactive session.

You can use the `.words()` method to get a list of stop words in English. To test the function, let us run it on our sample tweet. Add the following lines to the end of the `nlp_test.py` file:

```
nlp_test.py  
...  
from nltk.corpus import stopwords  
stop_words = stopwords.words('english')  
  
print(remove_noise(tweet_tokens[0], stop_words))
```

Copy

After saving and closing the file, run the script again to receive output similar to the following:

Output

```
['#followfriday', 'top', 'engage', 'member', 'community', 'week', ':)']
```

Notice that the function removes all `@` mentions, stop words, and converts the words to lowercase.

Before proceeding to the modeling exercise in the next step, use the `remove_noise()` function to clean the positive and negative tweets. Comment out the line to print the output of `remove_noise()` on the sample tweet and add the following to the `nlp_test.py` script:

```
nlp_test.py  
...  
from nltk.corpus import stopwords  
stop_words = stopwords.words('english')  
  
#print(remove_noise(tweet_tokens[0], stop_words))  
  
positive_tweet_tokens = twitter_samples.tokenized('positive_tweets.json')  
negative_tweet_tokens = twitter_samples.tokenized('negative_tweets.json')  
  
positive_cleaned_tokens_list = []  
negative_cleaned_tokens_list = []  
  
for tokens in positive_tweet_tokens:
```

```
    positive_cleaned_tokens_list.append(remove_noise(tokens, stop_words))

for tokens in negative_tweet_tokens:
    negative_cleaned_tokens_list.append(remove_noise(tokens, stop_words))
```

Now that you've added the code to clean the sample tweets, you may want to compare the original tokens to the cleaned tokens for a sample tweet. If you'd like to test this, add the following code to the file to compare both versions of the 500th tweet in the list:

```
nlp_test.py
...
print(positive_tweet_tokens[500])
print(positive_cleaned_tokens_list[500])
```

Copy

Save and close the file and run the script. From the output you will see that the punctuation and links have been removed, and the words have been converted to lowercase.

Output

```
['Dang', 'that', 'is', 'some', 'rad', '@AbzuGame', '#fanart', '!', ':D',
'https://t.co/bI8k8tb9ht']
['dang', 'rad', '#fanart', ':d']
```

There are certain issues that might arise during the preprocessing of text. For instance, words without spaces (“iLoveYou”) will be treated as one and it can be difficult to separate such words. Furthermore, “Hi”, “Hii”, and “Hiiii” will be treated differently by the script unless you write something specific to tackle the issue. It's common to fine tune the noise removal process for your specific data.

Now that you've seen the `remove_noise()` function in action, be sure to comment out or remove the last two lines from the script so you can add more to it:

```
nlp_test.py
...
#print(positive_tweet_tokens[500])
#print(positive_cleaned_tokens_list[500])
```

In this step you removed noise from the data to make the analysis more effective. In the next step you will analyze the data to find the most common words in your sample dataset.

## Step 5 — Determining Word Density

The most basic form of analysis on textual data is to take out the word frequency. A single tweet is too small of an entity to find out the distribution of words, hence, the analysis of the frequency of words would be done on all positive tweets.

The following snippet defines a generator function, named `get_all_words`, that takes a list of tweets as an argument to provide a list of words in all of the tweet tokens joined. Add the following code to your `nlp_test.py` file:

```
nlp_test.py  
...  
  
def get_all_words(cleaned_tokens_list):  
    for tokens in cleaned_tokens_list:  
        for token in tokens:  
            yield token  
  
all_pos_words = get_all_words(positive_cleaned_tokens_list)
```

Copy

Now that you have compiled all words in the sample of tweets, you can find out which are the most common words using the `FreqDist` class of NLTK. Adding the following code to the `nlp_test.py` file:

```
nlp_test.py  
from nltk import FreqDist  
  
freq_dist_pos = FreqDist(all_pos_words)  
print(freq_dist_pos.most_common(10))
```

Copy

The `.most_common()` method lists the words which occur most frequently in the data. Save and close the file after making these changes.

When you run the file now, you will find the most common terms in the data:

```
Output  
[(':', 3691),  
 (':-)', 701),  
 (':d', 658),  
 ('thanks', 388),  
 ('follow', 357),  
 ('love', 333),  
 ('...', 290),  
 ('good', 283),  
 ('get', 263),  
 ('thank', 253)]
```

From this data, you can see that emoticon entities form some of the most common parts of positive tweets. Before proceeding to the next step, make sure you comment out the last line of the script that prints the top ten tokens.

To summarize, you extracted the tweets from `nltk`, tokenized, normalized, and cleaned up the tweets for using in the model. Finally, you also looked at the frequencies of tokens in the data and checked the frequencies of the top ten tokens.

In the next step you will prepare data for sentiment analysis.

## Step 6 — Preparing Data for the Model

Sentiment analysis is a process of identifying an attitude of the author on a topic that is being written about. You will create a training data set to train a model. It is a supervised learning machine learning process, which requires you to associate each dataset with a “sentiment” for training. In this tutorial, your model will use the “positive” and “negative” sentiments.

Sentiment analysis can be used to categorize text into a variety of sentiments. For simplicity and availability of the training dataset, this tutorial helps you train your model in only two categories, positive and negative.

A model is a description of a system using rules and equations. It may be as simple as an equation which predicts the weight of a person, given their height. A sentiment analysis model that you will build would associate tweets with a positive or a negative sentiment. You will need to split your dataset into two parts. The

purpose of the first part is to build the model, whereas the next part tests the performance of the model.

In the data preparation step, you will prepare the data for sentiment analysis by converting tokens to the dictionary form and then split the data for training and testing purposes.

### Converting Tokens to a Dictionary

First, you will prepare the data to be fed into the model. You will use the Naive Bayes classifier in NLTK to perform the modeling exercise. Notice that the model requires not just a list of words in a tweet, but a Python dictionary with words as keys and `True` as values. The following function makes a generator function to change the format of the cleaned data.

Add the following code to convert the tweets from a list of cleaned tokens to dictionaries with keys as the tokens and `True` as values. The corresponding dictionaries are stored

in `positive_tokens_for_model` and `negative_tokens_for_model`.

```
nlp_test.py  
...  
def get_tweets_for_model(cleaned_tokens_list):  
    for tweet_tokens in cleaned_tokens_list:  
        yield dict([token, True] for token in tweet_tokens)  
  
positive_tokens_for_model = get_tweets_for_model(positive_cleaned_tokens_list)  
negative_tokens_for_model =  
get_tweets_for_model(negative_cleaned_tokens_list)
```

Copy

### Splitting the Dataset for Training and Testing the Model

Next, you need to prepare the data for training the `NaiveBayesClassifier` class. Add the following code to the file to prepare the data:

```
nlp_test.py  
...  
import random  
  
positive_dataset = [(tweet_dict, "Positive")  
    for tweet_dict in positive_tokens_for_model]
```

```
negative_dataset = [(tweet_dict, "Negative")
    for tweet_dict in negative_tokens_for_model]

dataset = positive_dataset + negative_dataset

random.shuffle(dataset)

train_data = dataset[:7000]
test_data = dataset[7000:]
```

Copy

This code attaches a Positive or Negative label to each tweet. It then creates a dataset by joining the positive and negative tweets.

By default, the data contains all positive tweets followed by all negative tweets in sequence. When training the model, you should provide a sample of your data that does not contain any bias. To avoid bias, you've added code to randomly arrange the data using the .shuffle() method of random.

Finally, the code splits the shuffled data into a ratio of 70:30 for training and testing, respectively. Since the number of tweets is 10000, you can use the first 7000 tweets from the shuffled dataset for training the model and the final 3000 for testing the model.

In this step, you converted the cleaned tokens to a dictionary form, randomly shuffled the dataset, and split it into training and testing data.

### Step 7 — Building and Testing the Model

Finally, you can use the NaiveBayesClassifier class to build the model. Use the .train() method to train the model and the .accuracy() method to test the model on the testing data.

```
nlp_test.py
...
from nltk import classify
from nltk import NaiveBayesClassifier
classifier = NaiveBayesClassifier.train(train_data)
```

```
print("Accuracy is:", classify.accuracy(classifier, test_data))
```

```
print(classifier.show_most_informative_features(10))
```

Copy

Save, close, and execute the file after adding the code. The output of the code will be as follows:

Output

```
Accuracy is: 0.9956666666666667
```

#### Most Informative Features

:( = True	Negati : Positi = 2085.6 : 1.0
:) = True	Positi : Negati = 986.0 : 1.0
welcome = True	Positi : Negati = 37.2 : 1.0
arrive = True	Positi : Negati = 31.3 : 1.0
sad = True	Negati : Positi = 25.9 : 1.0
follower = True	Positi : Negati = 21.1 : 1.0
bam = True	Positi : Negati = 20.7 : 1.0
glad = True	Positi : Negati = 18.1 : 1.0
x15 = True	Negati : Positi = 15.9 : 1.0
community = True	Positi : Negati = 14.1 : 1.0

Accuracy is defined as the percentage of tweets in the testing dataset for which the model was correctly able to predict the sentiment. A 99.5% accuracy on the test set is pretty good.

In the table that shows the most informative features, every row in the output shows the ratio of occurrence of a token in positive and negative tagged tweets in the training dataset. The first row in the data signifies that in all tweets containing the token :(, the ratio of negative to positives tweets was 2085.6 to 1. Interestingly, it seems that there was one token with :( in the positive datasets. You can see that the top two discriminating items in the text are the emoticons. Further, words such as sad lead to negative sentiments, whereas welcome and glad are associated with positive sentiments.

Next, you can check how the model performs on random tweets from Twitter. Add this code to the file:

```
nlp_test.py  
...  
from nltk.tokenize import word_tokenize  
  
custom_tweet = "I ordered just once from TerribleCo, they screwed up, never used  
the app again."  
  
custom_tokens = remove_noise(word_tokenize(custom_tweet))  
  
print(classifier.classify(dict([token, True] for token in custom_tokens)))
```

Copy

This code will allow you to test custom tweets by updating the string associated with the `custom_tweet` variable. Save and close the file after making these changes.

Run the script to analyze the custom text. Here is the output for the custom text in the example:

```
Output  
'Negative'
```

You can also check if it characterizes positive tweets correctly:

```
nlp_test.py  
...  
custom_tweet = 'Congrats #SportStar on your 7th best goal from last season  
winning goal of the year :) #Baller #Topbin #oneofmanyworldies'
```

Here is the output:

```
Output  
'Positive'
```

Now that you've tested both positive and negative sentiments, update the variable to test a more complex sentiment like sarcasm.

## nlp\_test.py

...

```
custom_tweet = "Thank you for sending my baggage to CityX and flying me to  
CityY at the same time. Brilliant service. #thanksGenericAirline"
```

Here is the output:

Output

'Positive'

The model classified this example as positive. This is because the training data wasn't comprehensive enough to classify sarcastic tweets as negative. In case you want your model to predict sarcasm, you would need to provide sufficient amount of training data to train it accordingly.

In this step you built and tested the model. You also explored some of its limitations, such as not detecting sarcasm in particular examples. Your completed code still has artifacts leftover from following the tutorial, so the next step will guide you through aligning the code to Python's best practices.

### Step 8 — Cleaning Up the Code (Optional)

Though you have completed the tutorial, it is recommended to reorganize the code in the `nlp_test.py` file to follow best programming practices. Per best practice, your code should meet this criteria:

- All imports should be at the top of the file. Imports from the same library should be grouped together in a single statement.
- All functions should be defined after the imports.
- All the statements in the file should be housed under an `if __name__ == "__main__":` condition. This ensures that the statements are not executed if you are importing the functions of the file in another file.

We will also remove the code that was commented out by following the tutorial, along with the `lemmatize_sentence` function, as the lemmatization is completed by the new `remove_noise` function.

Here is the cleaned version of `nlp_test.py`:

```
from nltk.stem.wordnet import WordNetLemmatizer  
from nltk.corpus import twitter_samples, stopwords  
from nltk.tag import pos_tag
```

```
from nltk.tokenize import word_tokenize
from nltk import FreqDist, classify, NaiveBayesClassifier

import re, string, random

def remove_noise(tweet_tokens, stop_words = ()):

    cleaned_tokens = []

    for token, tag in pos_tag(tweet_tokens):
        token = re.sub('http[s]?://(?:[a-zA-Z][0-9]|[$-_@.&+#!][!*\\(\\),]|"\n|(?:%[0-9a-fA-F][0-9a-fA-F]))+', "", token)
        token = re.sub("@[A-Za-z0-9_]+", "", token)

        if tag.startswith("NN"):
            pos = 'n'
        elif tag.startswith('VB'):
            pos = 'v'
        else:
            pos = 'a'

        lemmatizer = WordNetLemmatizer()
        token = lemmatizer.lemmatize(token, pos)

        if len(token) > 0 and token not in string.punctuation and token.lower() not in stop_words:
            cleaned_tokens.append(token.lower())

    return cleaned_tokens

def get_all_words(cleaned_tokens_list):
    for tokens in cleaned_tokens_list:
        for token in tokens:
            yield token

def get_tweets_for_model(cleaned_tokens_list):
```

```
for tweet_tokens in cleaned_tokens_list:
    yield dict([token, True] for token in tweet_tokens)

if __name__ == "__main__":
    positive_tweets = twitter_samples.strings('positive_tweets.json')
    negative_tweets = twitter_samples.strings('negative_tweets.json')
    text = twitter_samples.strings('tweets.20150430-223406.json')
    tweet_tokens = twitter_samples.tokenized('positive_tweets.json')[0]

    stop_words = stopwords.words('english')

    positive_tweet_tokens = twitter_samples.tokenized('positive_tweets.json')
    negative_tweet_tokens = twitter_samples.tokenized('negative_tweets.json')

    positive_cleaned_tokens_list = []
    negative_cleaned_tokens_list = []

    for tokens in positive_tweet_tokens:
        positive_cleaned_tokens_list.append(remove_noise(tokens, stop_words))

    for tokens in negative_tweet_tokens:
        negative_cleaned_tokens_list.append(remove_noise(tokens, stop_words))

    all_pos_words = get_all_words(positive_cleaned_tokens_list)

    freq_dist_pos = FreqDist(all_pos_words)
    print(freq_dist_pos.most_common(10))

    positive_tokens_for_model =
    get_tweets_for_model(positive_cleaned_tokens_list)
    negative_tokens_for_model =
    get_tweets_for_model(negative_cleaned_tokens_list)

    positive_dataset = [(tweet_dict, "Positive")]
```

```
for tweet_dict in positive_tokens_for_model]

negative_dataset = [(tweet_dict, "Negative")
                     for tweet_dict in negative_tokens_for_model]

dataset = positive_dataset + negative_dataset

random.shuffle(dataset)

train_data = dataset[:7000]
test_data = dataset[7000:]

classifier = NaiveBayesClassifier.train(train_data)

print("Accuracy is:", classify.accuracy(classifier, test_data))

print(classifier.show_most_informative_features(10))

custom_tweet = "I ordered just once from TerribleCo, they screwed up, never used the app again."

custom_tokens = remove_noise(word_tokenize(custom_tweet))

print(custom_tweet, classifier.classify(dict([token, True] for token in custom_tokens)))
```

Copy

## Conclusion

This tutorial introduced you to a basic sentiment analysis model using the `nltk` library in Python 3. First, you performed pre-processing on tweets by tokenizing a tweet, normalizing the words, and removing noise. Next, you visualized frequently occurring items in the data. Finally, you built a model to associate tweets to a particular sentiment.

A supervised learning model is only as good as its training data. To further strengthen the model, you could consider adding more categories like excitement and anger. In this tutorial, you have only scratched the surface by

building a rudimentary model. Here's a detailed guide on various considerations that one must take care of while performing sentiment analysis.