Libraries Used

In [1]:

```
import re
import nltk
from nltk import FreqDist
from nltk.classify import NaiveBayesClassifier
from nltk.classify.util import accuracy
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer
from collections import Counter
```

Data Import

After cloning the github repository, I am importing the tweets...

In [2]:

```
text_pos = []
labels_pos = []
with open("pos_tweets.txt") as f:
    for i in f:
        text_pos.append(i)
        labels_pos.append('pos')

text_neg = []
labels_neg = []
with open("neg_tweets.txt") as f:
    for i in f:
        text_neg.append(i)
        labels_neg.append('neg')
```

Preparing the Dataset

Creating a training and a testing dataset

In [3]:

```
pos_train, pos_test, pos_labels_train, pos_labels_test = train_test_split(text_pos, labels_pos, tes
t_size=0.20)
neg_train, neg_test, neg_labels_train, neg_labels_test = train_test_split(text_neg, labels_neg, tes
t_size=0.20)
```

In [4]:

```
training_tweets = pos_train + neg_train
training_labels = pos_labels_train + neg_labels_train

test_tweets = pos_test + neg_labels_test
test_labels = pos_labels_test + neg_labels_test
```

Create Important Feature list

Converting the tweets as a tuple of list of words (removing punctuation and keeping lowercase) and sentiment.

Note: I am not removing stopwords, as the number of words in tweet are very low, and by removing them we can remove a significant portion of a tweet, which would hamper the accuracy of the model.

In [5]:

```
#stop_words = set(stopwords.words('english'))

# Function to remove Punctuation and keep everything in lower case
def rem_Punct(sent):
    tokenizer = RegexpTokenizer(r'\w+')
    word_tokens = tokenizer.tokenize(sent)
    return([w.lower() for w in word_tokens])
    #return([w.lower() for w in word_tokens if not w.lower() in stop_words])
```

In [6]:

```
# Training
training_tweetslist = []
for i, tweet in enumerate(training_tweets):
    training_tweetslist.append((rem_Punct(tweet), training_labels[i]))

# Testing
test_tweetslist = []
for i, tweet in enumerate(test_tweets):
    test_tweetslist.append((rem_Punct(tweet), test_labels[i]))
```

In [7]:

```
# Getting a list of all words in the all tweets of the training dataset
def get_words_in_tweets(tweets):
    all_words = []
    for (words, sentiment) in tweets:
        all_words.extend(words)
    return all_words
```

In [8]:

```
# We keep 500 most common words in the training dataset as Word Features
allwords = Counter(get_words_in_tweets(training_tweetslist))
word_features = [word_count[0] for word_count in allwords.most_common(500)]
```

As we can see, these are the most common and useful words in tweets, therefore we use them to access the sentiment of each tweet !!

In [9]:

```
# Extracting the features list for each tweet
def extract_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features[word] = (word in document_words)
    return features
```

In [10]:

```
training_set = nltk.classify.apply_features(extract_features, training_tweetslist)
```

In [20]:

```
test_set = nltk.classify.apply_features(extract_features, test_tweetslist)
```

Building a Classifier

Now, that we have our training set, we will built a classifier

In [12]:

```
classifier = NaiveBayesClassifier.train(training_set)
```

In [13]:

```
classifier.show_most_informative_features()
```

```
Most Informative Features
                      no = True              neg : pos    =     25.7 : 1.0
                headache = True              neg : pos    =     18.2 : 1.0
                 awesome = True              pos : neg    =     16.6 : 1.0
                   thank = True              pos : neg    =     15.7 : 1.0
                 excited = True              pos : neg    =     15.7 : 1.0
                   great = True              pos : neg    =     14.2 : 1.0
                  follow = True              pos : neg    =     14.2 : 1.0
               beautiful = True              pos : neg    =     12.7 : 1.0
                    love = True              pos : neg    =     11.2 : 1.0
                    haha = True              pos : neg    =      9.2 : 1.0
```

Examples to test

In [14]:

```
example1 = "Twilio is an awesome company!"
print(classifier.classify(extract_features(example1.split())))
```

```
pos
```

In [16]:

```
example2 = "I'm sad that Twilio doesn't have even more blog posts!"
print(classifier.classify(extract_features(example2.split())))
```

```
neg
```

In [17]:

```
example3 = "I have no headache!"
print(classifier.classify(extract_features(example3.split())))
```

```
neg
```

Explanation : As we can see, the model choose the tweet to be negative even though it is positive, this is presence of more negative word such as "no", "headache" which have a high neg:pos ratio . Therefore it is classified as 'Negative' as the model is unable to take into account the context and language used in the tweets.

Accuracy of the test set

In [22]:

```
print("Accuracy of the test set: {}".format(accuracy(classifier, test_set)))
```

```
Accuracy of the test set: 0.917910447761194
```

In [3]:

```
#The accuracy of the test set is 91.79%
```

Testing on 10 random Tweets

In [32]:

```
example_pos_tweets = [
    "Coolest fan we’ve ever seen",
    "sh in a good mood .....tlk to me",
    "good morning I love you!!",
    "Thanks, I need all the help i can get.",
    "- @haugern The servers are now backup, if you experience any more problems then please let me
know  Sorry about the delay..."
]

example_neg_tweets = [
    'Donald Trump’s administration: “Government by the worst men.”',
    'Their lies are not just lies. Their lies are authoritarian propaganda.',
    "or i just worry too much?",
    "You're the only one who can see this cause no one else is following me this is for you becaus
e you're pretty awesome",
    "Very sad about Iran."
]
```

In [34]:

```
for tweet in example_pos_tweets:
    print(tweet)
    print("Classification: {}".format(classifier.classify(extract_features(tweet.split()))))
    print()
```

```
Coolest fan we’ve ever seen
Classification: pos

sh in a good mood .....tlk to me
Classification: pos

good morning I love you!!
Classification: pos

Thanks, I need all the help i can get.
Classification: neg

- @haugern The servers are now backup, if you experience any more problems then please let me know
Sorry about the delay...
Classification: pos
```

Explanation :

As we can see, the model choose 1 tweet out of 5 (4th tweet) to be Negative even though all the tweets are positive, this is due to the words such as "need", "help" mostly these words are used in negative context, and the model does not account for the languages of the sentence therefore, it classified as negative.

In [35]:

```
for tweet in example_neg_tweets:
    print(tweet)
    print("Classification: {}".format(classifier.classify(extract_features(tweet.split()))))

    print()
```

```
Donald Trump’s administration: “Government by the worst men.”
Classification: neg

Their lies are not just lies. Their lies are authoritarian propaganda.
Classification: neg

or i just worry too much?
Classification: neg

You're the only one who can see this cause no one else is following me this is for you because you
're pretty awesome
Classification: neg
```

```
Very sad about Iran.
Classification: neg
```

Note ::

Even through the nature of 4th tweet can be considered as positive, the model detects it as negative due to presence of more higher weightage words such as 'no' etc when compared to lower weightage positive words such as "awesome". Also, My model is able to detect all tweet correctly, which proves the model is working well, with high accuracy.

Steps to Improve Accuracy:

1. Currently we have only removed punctuation, we can look at other NLP elements such as Stemming to filter words such as "abend", "abending", "abended" into one word "abend".
2. Naive Bayes does not consider relationship between words, but other classifiers such as ensemble trees can be used to check if we can obtain better results.
3. I am only keeping 500 most common words, but more words can be utilized to improve the model.
4. Increasing the corpus to include more labelled tweets will help accuracy of the model.

Useful Links

1. https://www.twilio.com/blog/2017/09/sentiment-analysis-python-messy-data-nltk.html
2. https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6
3. https://www.laurentluce.com/posts/twitter-sentiment-analysis-using-python-and-nltk/
4. 10 random tweet examples :: https://www.kaggle.com/c/twitter-sentiment-analysis2