CIS 9660 - Data Mining for Business Analytics

Twitter Sentiment Analysis

Macrin Francis
David Arizmendi
Stylianos Markou
Juliocarlos Velez

Motivation for Project idea:
Sentiment analysis has been widely adopted by several organizations around the world. Being able to see the sentiment behind anything enables organizations to better strategize and plan for the future. For example, sentiment analysis is extremely useful in social media monitoring. Companies can gain important insights by understanding the wider public opinion on the products or services they offer.

Description of the issues or opportunities the project will address:
We find sentiment analysis fascinating. For that reason, we decided that for our final project, we wanted to perform sentiment analysis on the City University of New York (CUNY). We decided to pull our data from two different sources. The first source was directly from Twitter. We had to apply for Twitter developer accounts and wait for approval before having access to Twitter's data. Once we got approved, we retrieved a maximum number of 15,000 tweets using Tweepy. The second source was NLTK. We used the NLTK module to pull a sample of tweets (10,000 to be precise). Each observation in the data is a tweet. Sample data from NLTK was of special importance because it added data that had been already labeled correctly, which would be useful when combining our two datasets (the collection of tweets and the NLTK sample of tweets).

Code:
Loading the packages :

We started with loading the necessary packages for performing the sentiment analysis. We are using the NLTK package , "punkt", "wordnet", "averaged_perception_tagger", "stopwords", "twitter_samples", "part of speech tag", "FreqDist", "random".

Reading in Datasets:
From NLTK Library :

We created two datasets from NLTK that contain various tweets to train and test the model.

```
# Create variables for two datasets from NLTK that contain various tweets to train and test the model
positive_tweets = twitter_samples.strings('positive_tweets.json')
negative_tweets = twitter_samples.strings('negative_tweets.json')
```

Proceeded to make the positive and negative tweets json into a separate data frame, add labels , merge them together and shuffle them.

```
NLTK_tweets_df.head()
```

	tweet	label
0	AlyssaC_HK I use Chrome :)	1
1	Michael37311757 @royjohnwatts @LBC @darrenadam...	1
2	matteomeacci Well, they say Europe is on the r...	1
3	jaimeemelanie_ lmaoo! The best songs ever <...	1
4	LouiseMillerx @RedShoes4Life @x_Kisaragi @Dayv...	0

Using Twitter API to pull the tweets:

Using the account details from the developer account , we are defining a function for pulling tweets from Twitter in a dataframe format.

Sample:

```
class TwitterClient(object):
    def __init__(self):
        #Initialization method.
        try:
            # create OAuthHandler object
            auth = OAuthHandler(consumer_key, consumer_secret)
            # set access token and secret
            auth.set_access_token(access_token, access_token_secret)
            # create tweepy API object to fetch tweets
            # add hyper parameter 'proxy' if executing from behind proxy "proxy='http://172.22.218.218:8085'"
            self.api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)

        except tweepy.TweepError as e:
            print(f"Error: Tweeter Authentication Failed - \n{str(e)}")
```

We used “CUNY” as a keyword to pull tweets based on them and limited the number of tweets to 15000.

```
twitter_client = TwitterClient()
# calling function to get tweets
pulled_tweets_df = twitter_client.get_tweets('CUNY', maxTweets=15000)
print(f'tweets_df Shape - {pulled_tweets_df.shape}')
pulled_tweets_df.head(10)
```

Defining the fetch_sentiment_using_textblob and assigning the sentiments as 0 and 1 based on polarity.

```
def fetch_sentiment_using_textblob(text):
    analysis = TextBlob(text)
    return 1 if analysis.sentiment.polarity >= 0 else 0
```

Combining Dataset:

```
combined_tweet_df = pd.concat( [pulled_tweets_df, NLTK_tweets_df], ignore_index=True)
# shuffle your dataframe in-place and reset the index
combined_tweet_df = combined_tweet_df.sample(frac=1).reset_index(drop=True)
tweets_df = combined_tweet_df
tweets_df
```

Data Cleaning:

We cleaned the noises in the dataset by removing the @names, blank spaces, “RT”, filtering out the words that contain links like “https” and resetting the index. The cleaned data is then categorized into two columns, “tweets” and “label” and a final dataset is made.

```
#Creating dataframe
tweets_final_df = pd.DataFrame()
tweets_final_df['tweet'] = tweets_df['absolute_tidy_tweets']
tweets_final_df['label'] = tweets_df['label']
tweets_final_df
```

We also added Tweet Length and Word Count.

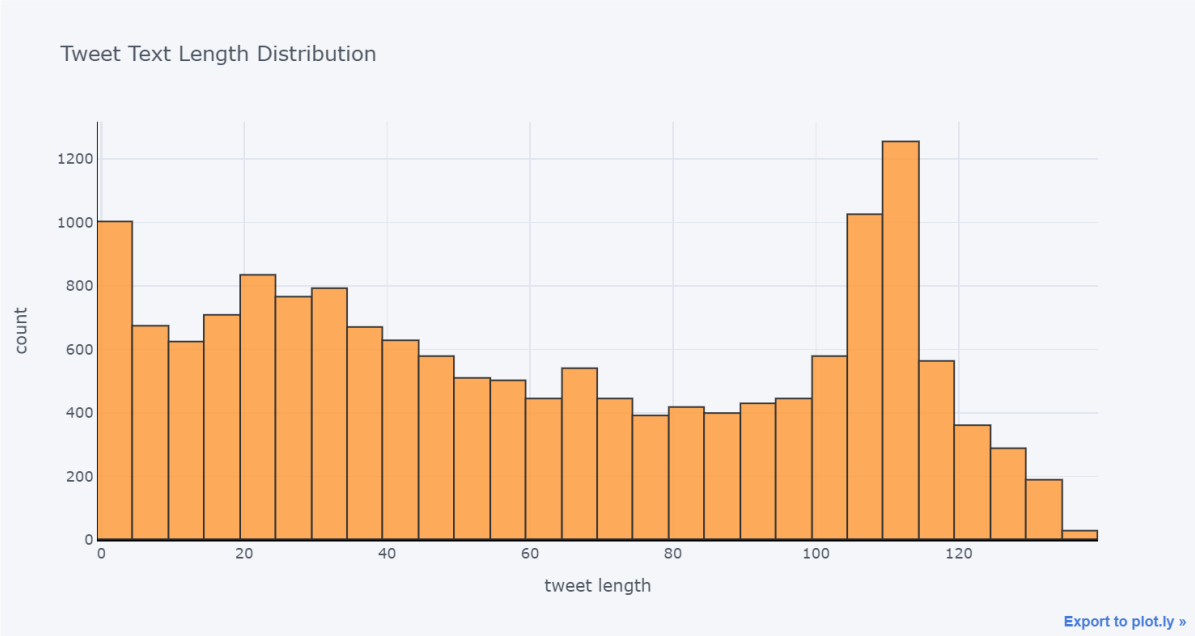
```
#Adding Tweet length and Word Count
tweets_final_df['Tweet_Length'] = tweets_final_df['tweet'].astype(str).apply(len)
tweets_final_df['Word_Count']=tweets_final_df['tweet'].apply(lambda x: len(str(x).split()))
tweets_final_df
```

	tweet	label	Tweet_Length	Word_Count
0	ust a bit to big to scan so will add them to t...	1	96	22
1	ast night was one of the worst night I pretty...	0	81	16
2	Today PM Historian Elizabeth Heath of Baruch...	1	109	15
3		1	6	0
4	o one is brave enough to watch all my snapchat...	0	53	11

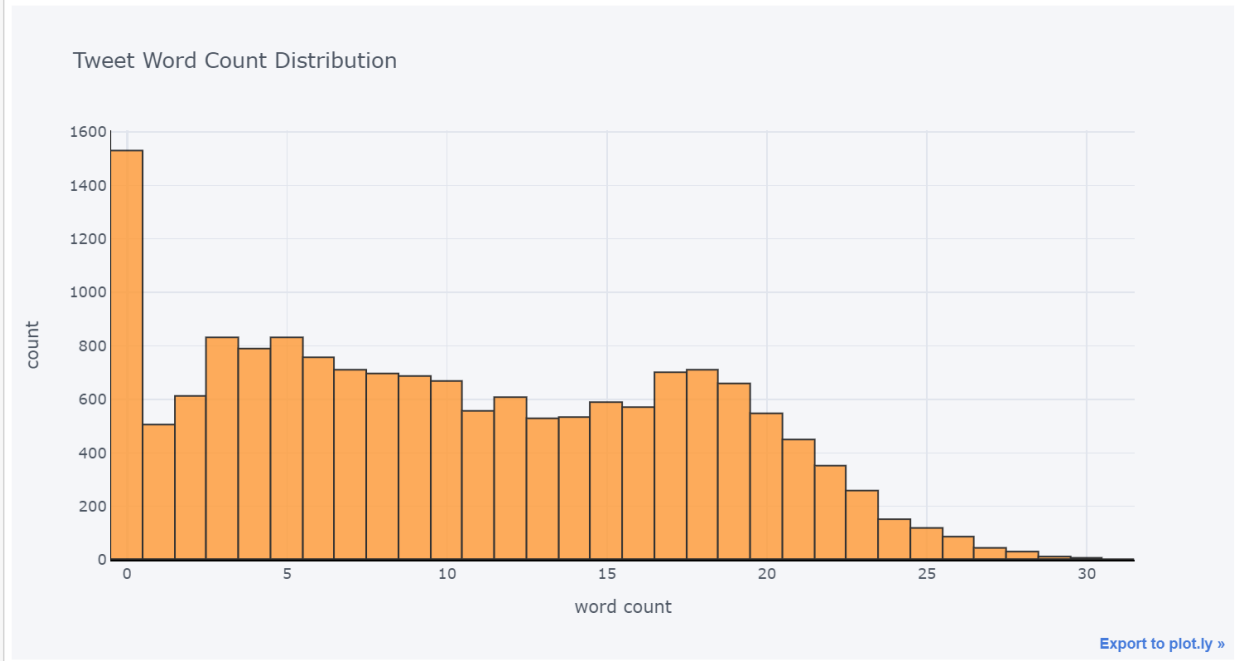
Data Visualization:

We then proceeded to do a few data visualizations to get an idea of how the dataset looked.

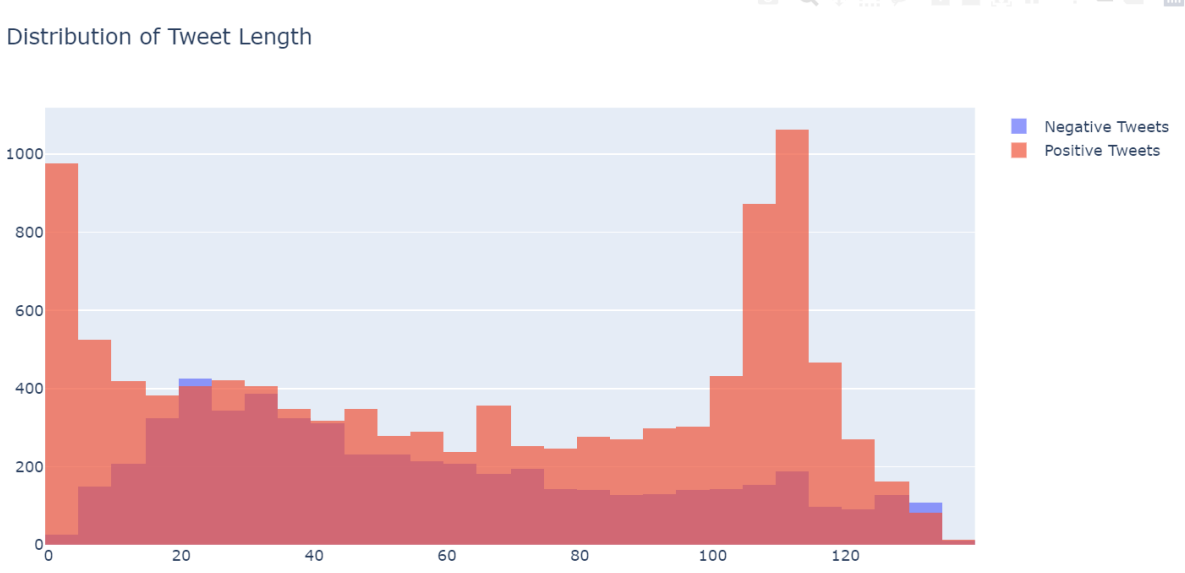
Tweet Text Length Distribution:



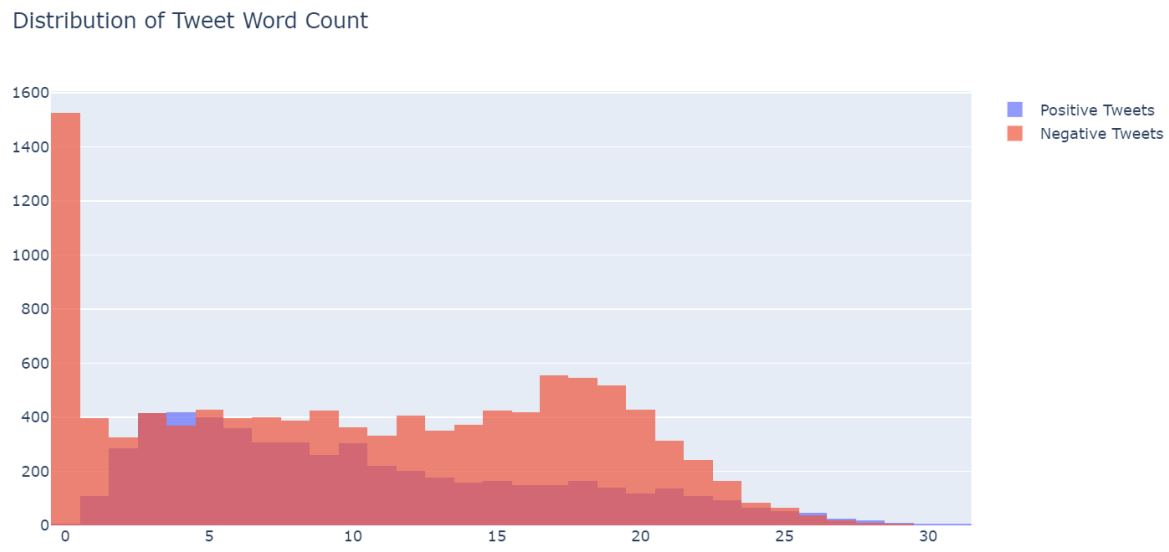
Tweet Word Count Distribution:



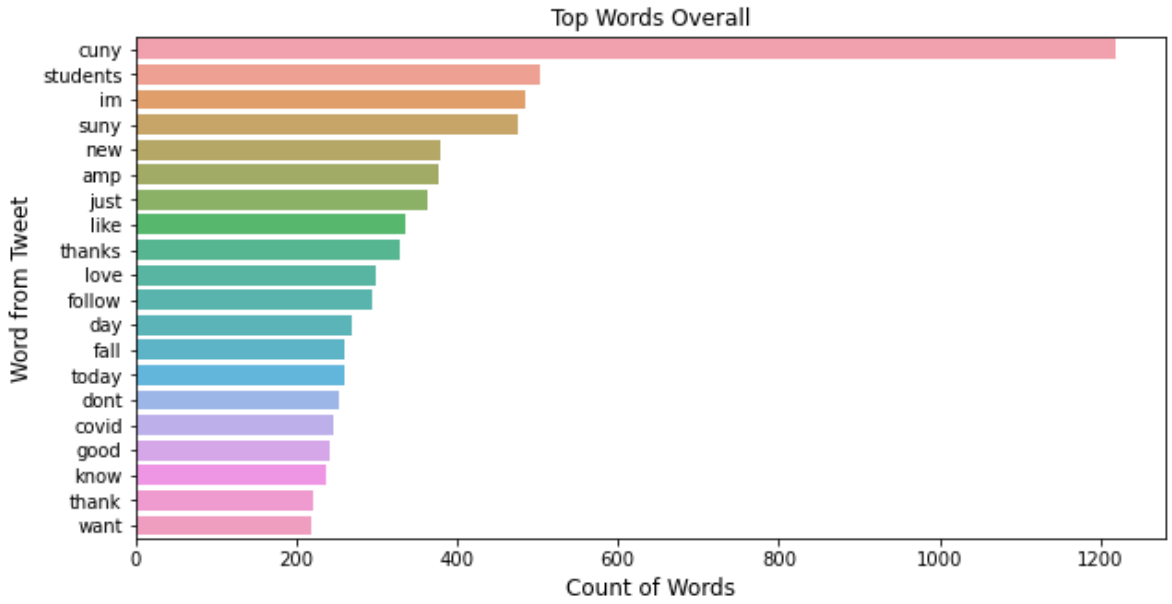
Distribution of Tweet Length:



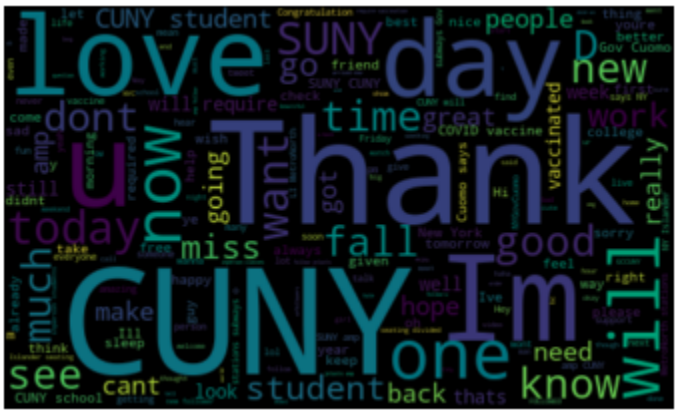
Distribution of Tweet Word Count:



Frequently Occuring Words:



Word Cloud:



Splitting the dataset into train and validation set

```
x = tweets_final_df.tweet
y = tweets_final_df.label
x_train, x_val, y_train, y_val = train_test_split(x, y, train_size = 0.7, random_state = 0)
```

Data Modeling:

Bag of Words

We are representing the tweets using a bag-of-words model and analysing them .

```
count_vec = CountVectorizer(ngram_range = (1,1), stop_words = "english", lowercase = True)
```

```
train_counts = count_vec.fit_transform(x_train)

bag_train = pd.DataFrame(train_counts.toarray(), columns=count_vec.get_feature_names())
```

```
word_counts = bag_train.sum()
word_counts = word_counts.sort_values(ascending = False)
word_counts.head(30)
```

We are normalizing with the word count of 20, 30, and 40 and are measuring their Area Under the Curve (AUC) for both training and validation set.

```
reduced_bag_train = bag_train[word_counts[word_counts >= 20].index]
reduced_bag_train = (reduced_bag_train - reduced_bag_train.mean()) / reduced_bag_train.std()
val_counts = count_vec.transform(x_val)

bag_val = pd.DataFrame(val_counts.toarray(), columns=count_vec.get_feature_names())
reduced_bag_val = bag_val[reduced_bag_train.columns]
reduced_bag_val = (reduced_bag_val - reduced_bag_val.mean()) / reduced_bag_val.std()
```

Model 1:

Logistic Regression

```
logit_model = LogisticRegression(random_state = 123, max_iter= 1000).fit(reduced_bag_train, y_train)
```

The AUC for training and validation sets are listed below.

AUC of training

```
fpr, tpr, thresholds = metrics.roc_curve(y_train, y_train_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.8388299881212035
```

```
y_val_prob = logit_model.predict_proba(reduced_bag_val)
```

AUC of validation

```
fpr, tpr, thresholds = metrics.roc_curve(y_val, y_val_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.8013857381255967
```

Model 2:

Grid Search:

We are using gridsearch to compute the optimum values of the parameters. We ran the model with the word count of 20.

```
# a wordcount of 20
reduced_bag_train = bag_train[word_counts[word_counts >= 20].index]
reduced_bag_train = (reduced_bag_train - reduced_bag_train.mean()) / reduced_bag_train.std()
val_counts = count_vec.transform(x_val)

bag_val = pd.DataFrame(val_counts.toarray(), columns=count_vec.get_feature_names())
reduced_bag_val = bag_val[reduced_bag_train.columns]
reduced_bag_val = (reduced_bag_val - reduced_bag_val.mean()) / reduced_bag_val.std()
```

The AUC for training and validation sets are listed below.

AUC for training set

```
y_train_prob = clf.predict_proba(reduced_tf_idf_train)
fpr, tpr, thresholds = metrics.roc_curve(y_train, y_train_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.6308771957421361
```

AUC for validation set

```
y_val_prob = clf.predict_proba(reduced_tf_idf_val)
fpr, tpr, thresholds = metrics.roc_curve(y_val, y_val_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.6164294151445715
```

Model 3:

Random Forest

We ran the model with a sample of 20 and 30 words and we found 20 words performed best.

```
# doing random forest with 20 words
parameters = {"max_depth":range(2, 8), "min_samples_leaf": range(5, 55, 5), "min_samples_split": range(10, 110, 5),
              "max_samples":[0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4], "max_features": [2, 3, 4, 5, 6],
              "n_estimators": [100, 150, 200, 250, 300, 350, 400]}
clf = RandomizedSearchCV(RandomForestClassifier(), parameters, n_jobs=4, scoring = "roc_auc", n_iter = 200,
                         random_state = 0)

clf.fit(reduced_bag_train, y_train)
```

```
RandomizedSearchCV(estimator=RandomForestClassifier(), n_iter=200, n_jobs=4,
                   param_distributions={'max_depth': range(2, 8),
                                        'max_features': [2, 3, 4, 5, 6],
                                        'max_samples': [0.1, 0.15, 0.2, 0.25,
                                                        0.3, 0.35, 0.4],
                                        'min_samples_leaf': range(5, 55, 5),
                                        'min_samples_split': range(10, 110, 5),
                                        'n_estimators': [100, 150, 200, 250,
                                                         300, 350, 400]},
                   random_state=0, scoring='roc_auc')
```

The AUC for training and validation sets are listed below.

AUC for training set

```
y_train_prob = clf.predict_proba(reduced_bag_train)
fpr, tpr, thresholds = metrics.roc_curve(y_train, y_train_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.8095366486509472
```

AUC for validation set

```
y_val_prob = clf.predict_proba(reduced_bag_val)
fpr, tpr, thresholds = metrics.roc_curve(y_val, y_val_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.7905138879032818
```

Model 4:

XGBoost:

We ran the model with a sample of 20 and 30 words and the AUC for training and validation sets are listed below for 20 words.

AUC for training set

```
y_train_prob = clf.predict_proba(reduced_bag_train)
fpr, tpr, thresholds = metrics.roc_curve(y_train, y_train_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.8335128297177633
```

AUC for validation set

```
y_val_prob = clf.predict_proba(reduced_bag_val)
fpr, tpr, thresholds = metrics.roc_curve(y_val, y_val_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.8083510551845857
```

We can clearly see the word counts with 20 performs better in all the models.

TF - IDF

We are using TF- IDF to evaluate how relevant a word is to a tweet. We repeated the same steps like we did for Bag of Words.

```
tf_idf = TfidfVectorizer(ngram_range = (1,1), stop_words = "english", lowercase = True)
```

```
train_tf_idf = tf_idf.fit_transform(x_train)

train_tf_idf = pd.DataFrame(train_tf_idf.toarray(), columns = tf_idf.get_feature_names())
```

Model 1
Logistic Regression:

AUC for training set

```
y_train_prob = logit_model.predict_proba(reduced_tf_idf_train)
fpr, tpr, thresholds = metrics.roc_curve(y_train, y_train_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.8419820746391793
```

AUC for validation set

```
y_val_prob = logit_model.predict_proba(reduced_tf_idf_val)
fpr, tpr, thresholds = metrics.roc_curve(y_val, y_val_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.8040573691997659
```

Model 2:

Grid Search:

AUC for training set

```
y_train_prob = clf.predict_proba(reduced_tf_idf_train)
fpr, tpr, thresholds = metrics.roc_curve(y_train, y_train_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.6308771957421361
```

AUC for validation set

```
y_val_prob = clf.predict_proba(reduced_tf_idf_val)
fpr, tpr, thresholds = metrics.roc_curve(y_val, y_val_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.6164294151445715
```

Model 3:

Random Forest:

AUC for training set

```
y_train_prob = clf.predict_proba(reduced_tf_idf_train)
fpr, tpr, thresholds = metrics.roc_curve(y_train, y_train_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.8214980104560414
```

AUC for validation set

```
y_val_prob = clf.predict_proba(reduced_tf_idf_val)
fpr, tpr, thresholds = metrics.roc_curve(y_val, y_val_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.7903914182746051
```

Model 4:

XGBoost:

AUC for training set

```
y_train_prob = clf.predict_proba(reduced_tf_idf_train)
fpr, tpr, thresholds = metrics.roc_curve(y_train, y_train_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.8577371920099903
```

AUC for validation set

```
y_val_prob = clf.predict_proba(reduced_tf_idf_val)
fpr, tpr, thresholds = metrics.roc_curve(y_val, y_val_prob[:,1], pos_label = 1)
metrics.auc(fpr, tpr)
```

```
0.7906995953542424
```

Using NLTK to Normalize, Remove Noise and Tokenize Tweets:

We then defined a function to normalize a sentence for which we have to generate the tags for each token in the text and then lemmatize each word using the position tag of each token of a tweet.

```
def lemmatize_sentence(tokens):
    lemmatizer = WordNetLemmatizer()
    lemmatized_sentence = []
    # 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
    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]))
```

Functions also removed the @ mentions and converted the words to lowercase.

```
def remove_noise(tweet_tokens, stop_words = ()):

    cleaned_tokens = []

    # 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.
    # Arguments: the tweet tokens and the tuple of stop words
    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'
        else:
            pos = 'a'
```

By using remove_noise() function, the positive and negative tweets were cleaned.

```
# Use the remove_noise() function to clean the positive and negative tweets.
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))
```

We proceeded to define a generator function that can take a list of tweets as an argument to provide a list of words in all of the tweet tokens joined.

```
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)
```

Splitting the datasets:
We then split the datasets into a ratio of 70:30 for training and testing models.

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

Data Modeling:

Loaded the packages needed for modeling like Random Forest Classifier, Gradient Boosting Classifier, Decision Tree Classifier, plot tree, Grid Search CV, Randomized Search CV.

```
import sklearn
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
import pandas as pd
import time
import numpy as np
import xgboost as xgb
```

Created dataframes with the positive_dataset and negative_dataset , created Label and concatenated everything as clean_tweet_df and shuffled it so as to reset the index.

```
# shuffle your dataframe in-place and reset the index
clean_tweet_df = clean_tweet_df.sample(frac=1).reset_index(drop=True)
clean_tweet_df = clean_tweet_df.fillna(0)
clean_tweet_df = clean_tweet_df.astype(int)
clean_tweet_df
```

We then split the entire dataset using train_test_split and then split the label out from both the sets.

```
# split the label out from the train and validation sets
clean_y_train_df = clean_train_data.Label
clean_x_train_df = clean_train_data.drop(columns = ["Label"])

clean_y_val_df = clean_val_data.Label
clean_x_val_df = clean_val_data.drop(columns = ["Label"])
```

Modeling :

We have modeled our datasets using various models.

Model 1:
NaiveBayes Classifier from NLTK:

```
#Building and Testing the Model

classifier = NaiveBayesClassifier.train(train_data_clean)

# Accuracy is defined as the percentage of tweets in the testing dataset
#for which the model was correctly able to predict the sentiment.
print("Accuracy (percent):", classify.accuracy(classifier, test_data_clean)*100)

# 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
classifier.show_most_informative_features(10)
```

```
Accuracy (percent): 99.66666666666667
Most Informative Features
                    :( = True           Negati : Positi =   2067.0 : 1.0
                    :) = True           Positi : Negati =    984.2 : 1.0
                   bam = True           Positi : Negati =     21.0 : 1.0
                  glad = True           Positi : Negati =     20.3 : 1.0
              follower = True           Positi : Negati =     19.4 : 1.0
                   sad = True           Negati : Positi =     18.6 : 1.0
                   x15 = True           Negati : Positi =     17.7 : 1.0
              followed = True           Negati : Positi =     15.4 : 1.0
             community = True           Positi : Negati =     13.7 : 1.0
                   ugh = True           Negati : Positi =     13.7 : 1.0
```

Using this model, we can see that each feature makes an independent and equal contribution to the outcome. We achieved 99.66 accuracy using this model.

Model 2:
Classifiers for NLTK from Sklearn:

We used a few classifiers like Original Naive Bayes , MNB Classifier, Bernoulli Classifier, Logistic Regression Classifier, SGD Classifier, SVC Classifier, Linear SVC Classifier and NuSVC Classifier.

```
Original Naive Bayes Algo accuracy percent: 99.66666666666667
MNB_classifier accuracy percent: 99.8
BernoulliNB_classifier accuracy percent: 99.76666666666667
LogisticRegression_classifier accuracy percent: 99.6
SGDClassifier_classifier accuracy percent: 99.63333333333333
SVC_classifier accuracy percent: 99.73333333333333
LinearSVC_classifier accuracy percent: 99.8
NuSVC_classifier accuracy percent: 98.6
```

We can see almost all classifiers had an accuracy percent ranging from 98.6 - 99.8

Model 3:
Bernoulli NB Classifier:

Among all the classifiers , we found BernoulliNB_ classifier has been the most consistent and performed best .

```
custom_tweet = "This is a test and i love this class."

custom_tokens = remove_noise(word_tokenize(custom_tweet))

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

```
Positive
```

```
custom_tweet = "This is a test and I dont know how i feel this class."

custom_tokens = remove_noise(word_tokenize(custom_tweet))

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

```
Negative
```