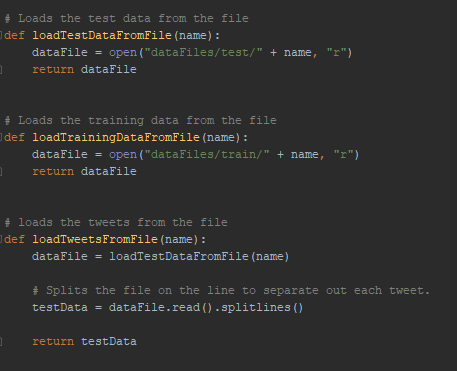
**Programming for Data Analytics Project B**

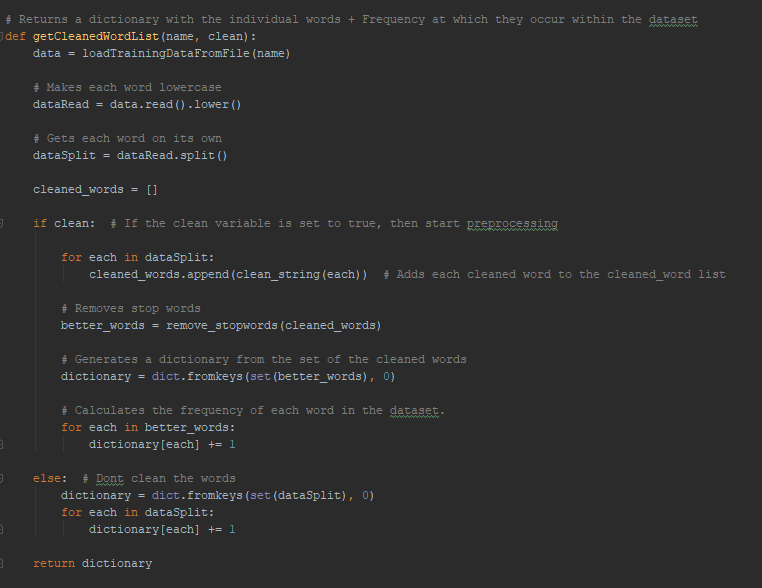
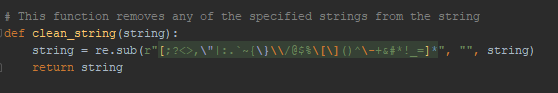
**Introduction**

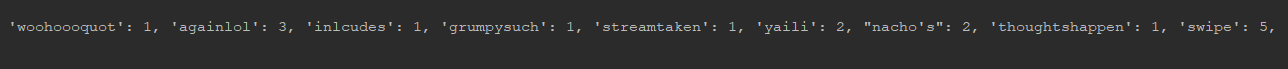
This is a project on tweet analysis using multinomial naïve-bayes classification learning algorithm. Sample data of both 400,000 positive and negative training tweets were processed and had their erroneous characters and stop-words removed. The probability of each word in the training data is determined and stored, and then checked against two test data files. These two files contained one-thousand positive and one-thousand negative tweets respectively, each tweet was then processed and checked against the training data to determine the positive or negative bias of a tweet.

**Stage One – Vocabulary Composition and Word Frequency Calculations:**

* The data for each file is loaded in through these specific functions, the functions load in either the training or test data, depending on when they are needed in the file.

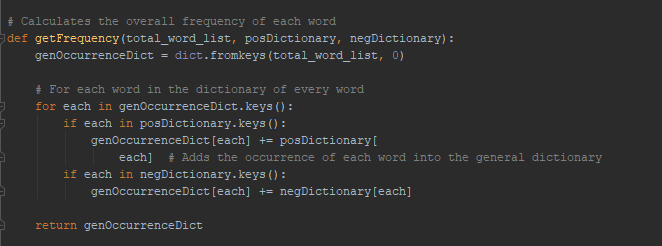


* The training data is then loaded in and split into individual words and then cleaned by use of the clean\_string function shown below. Stop words are removed in the remove\_stop\_words function, and then added into the better\_words list. A dictionary is then created from a set of better\_words, this is done to remove any duplicate occurrences of a word, as a set can only hold unique words.   
  Each word in the dictionary is then checked against the better\_words list for occurrences, and its corresponding integer value in the dictionary is incremented. 
* The data is cleaned by use of this function using the regex library, which removes any of the specified characters from the string
* The output of this looks like so

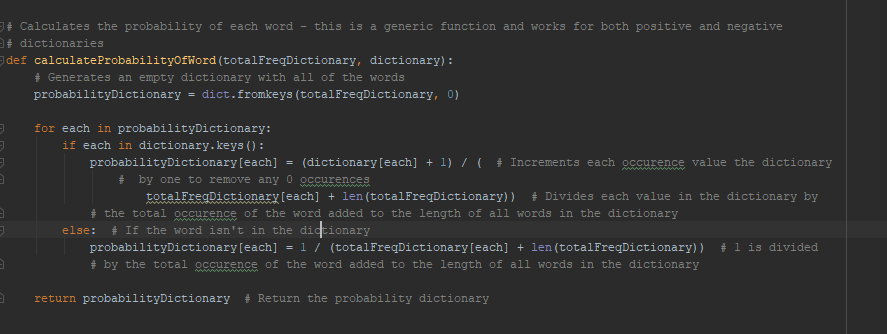


**Stage 2- Calculate Word Probability Calculations:**

* To calculate the probability of a word, the overall frequency of the word occurring in both files must be determined. This is done with the getFrequency function shown below. Each word in the generalOccurenceDictionary is checked against the words in the positive and negative dictionaries, if it occurs in either, the value in the generalOccuenceDictionary is incremented.

****

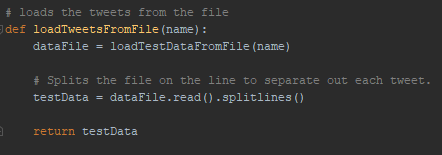
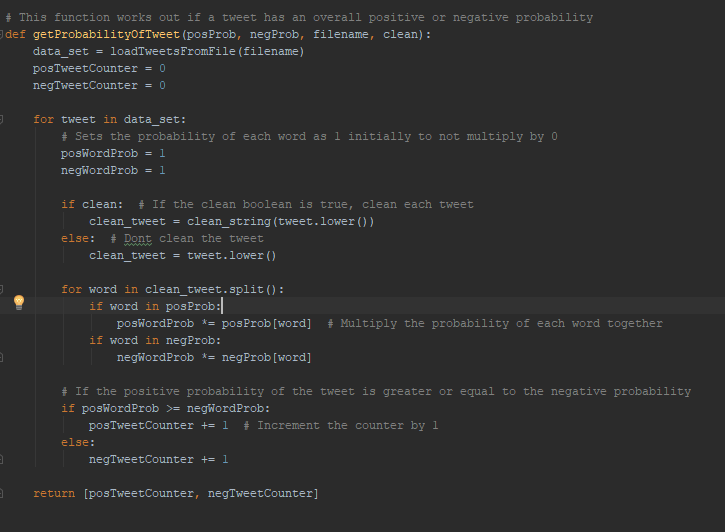
Now that the total frequency has been attained we can work out the probability of the word. This is done using the function below. For each word both the positive and negative values are worked out. The function shown below is a generic function which means that it is called twice, with either the positive or negative dictionary being passed in.



* The result of this function is as shown

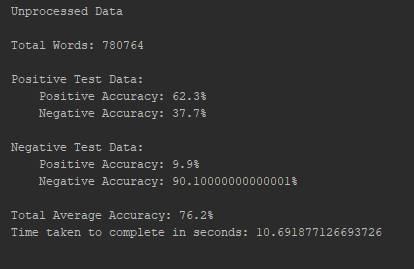
****

**Stage 3 – Classifying Unseen Tweets and Performing Basic Evaluation**

* Tweets are loaded in via the loadTweetFromFile function and split on the line, as each tweet is stored on a separate line. 
* The tweets are then loaded into the getProbabilityOfTweet function, and each tweet is set as lower case and cleaned using the same regex function as seen above, to ensure that both the training and test dataset use words without symbols. Stopwords do not need to be removed from the tweet as each word is checked against the dictionary that contains all the words found in the training sets, if a stop word is found in the tweet and is not found in the overall word dictionary, then its probability is ignored. 
* The probability of each word is multiplied against each of the other words in the tweet, and the overall probability is deduced. If the overall positive probability of the word is greater than or equal to the negative probability of the word, then the tweet is classified as positive, and vice versa.

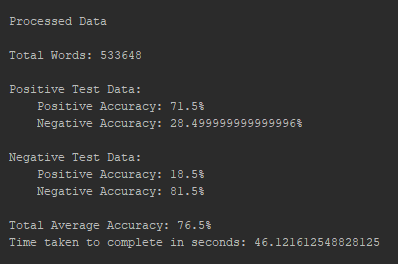
**Unprocessed Results**

* This is the results of the probability calculations of all tweets with an unprocessed training dataset and unprocessed tweet dataset. Note that the number of unique words is over 780,000.

****

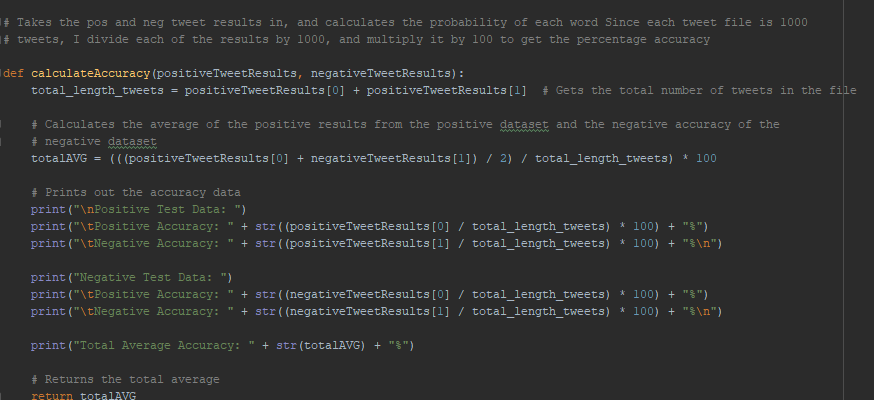
**Processed Results**

* The processed results give a different yield of data, with the positive dataset accuracy increasing by almost 10%, and the negative accuracy of the negative dataset decreasing by a similar margin. Through a brief look at the output of both the negative and positive datasets, it could be assumed that there were more stop words in the negative training file, these are words with no real positive or negative probability, and when their probability was worked out, was skewed negatively. The processed results also took 46 seconds to complete, which was a great deal longer than the unprocessed results which clocked in at 10 seconds. This is due to iteratively removing stop words and regexing unwanted symbols.



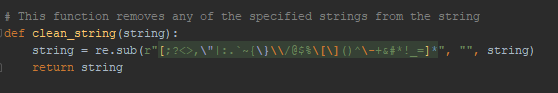
* **Accuracy Calculation**

The accuracy calculation is shown here and is explained by the comments in the code.

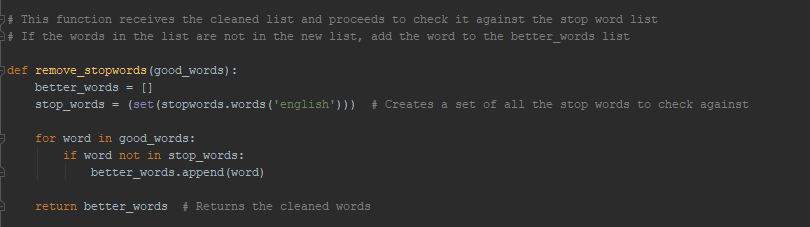
* ****

**Stage 4 – Additional Elements**

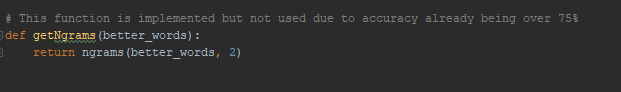
* Two major forms of pre-processing were used in the program. Regex and Stop word removal.
* This is the regex function, which removes any of the erroneous characters and leaves the words intact.

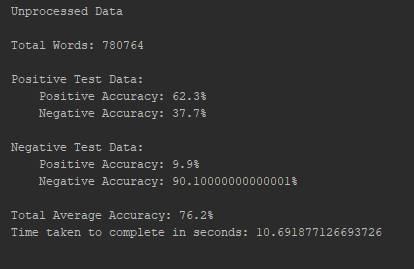


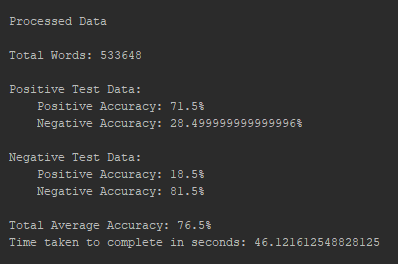
* This is the stop word removal function, where a list of regexed words were passed in and checked for occurrences of stop words. A new list is created with no stop words included.



* Included here is a function to create ngrams of 2 words next to eachother, but I was not sure how to implement ngram checking to increase the accuracy.

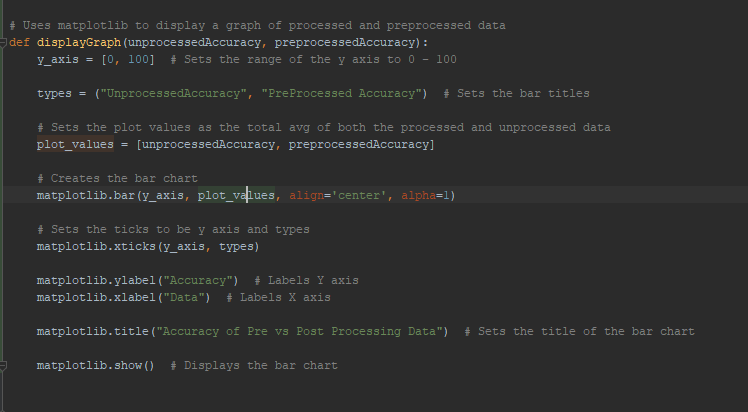


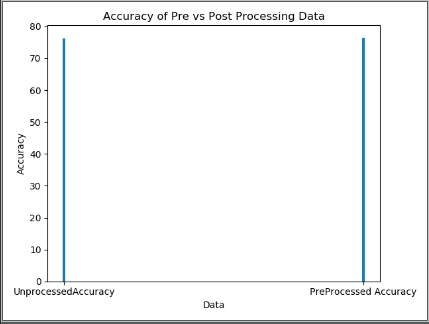
* As shown above in stage 4, there was almost no change in the average total accuracy, but the real accuracy of the training data was improved dramatically by close to 10% in each data set due to the stop word removal.
* 



**5. Visualization**

* Below is the function and graph created using matplotlib, it displays the average unprocessed accuracy and the average processed accuracy.

****

****