As a part of this assignment the course work was classifying the overall sentiment of a tweet as positive, negative or neutral. So initially all the required packages like sklearn, nltk, pandas etc. are downloaded and the respective classifiers are imported form the packages i.e. MultinominalNB, LogisticRegression etc. All the other components like stop words, TfidfTransformer, Pipeline are imported from their respective packages.

**Data Cleaning:**

The next task here is to preprocess the tweets. So, for that purpose the stop words are defined at first. The motive here is to access the stop words from nltk and to cease analysis on the tweets when we find words in tweet like is, a, uff, hmmm, uhh etc. However, using a generic list of stopwords can have a negative impact on sentiment analysis performance. Removing some common stopwords like "no", "not" etc. can change sentiment of a sentence, so we define some of these words under the variable white\_list\_words. Further, a for loop is used to remove the whitelisted words from nltk based stop\_words list, if it contains them. Then various preprocessing function are defined. The functions are as follows:  
a) function repeat\_char\_repl(tweets): The purpose of the function is to replace the characters in tweets with repeating characters for ex nooooo gets replaced with no and so on in the tweets.

b) function token(tweets): The purpose of the function is to use the nltk package feature word\_tokenize to tokenize the words in the tweets.

c) function special\_replace(tweets): The purpose of the function is to replace the special characters from the tweet like @, \*, # etc in the tweets.

d) function non\_alpha\_num\_replace(tweets): The purpose of the function is to replace all the non-alphanumeric characters except space in the tweets

e) function num\_replace(tweets): The purpose of the function is to remove numbers completely made of digits in the tweets.

f) function space\_replace(tweets): The purpose of the function is to remove extra spaces from the tweets.

g) function URL\_replace(tweets): The purpose of the function is to remove any possible URL in any format from the tweets.

h) function preprocessing(tweets): The purpose of the function is to call the other preprocessing functions in a specific order to facilitate the proper preprocessing of the tweets. It first converts the tweets into lower case, then calls URL\_replace() function, followed by special\_replace(), repeat\_char\_repl(), non\_alpha\_num\_replace(), num\_replace(), space\_replace(), and token().

**Feature extraction**

To carry out the feature extraction which is the second step of twitter sentiment analysis, I have used the ngrams feature i.e. unigrams and bigrams feature. For extracting the features, I have made use of scikit based CountVectorizer, which is used to convert a collection of text documents to a matrix of token counts. This implementation produces a sparse representation of the counts using scipy.sparse.csr\_matrix. The CountVectorizer has been used as shown below:  
 *CountVectorizer(min\_df=3, ngram\_range=(1,2)))*

The parameter, min\_df=3 in the above line means that when building the vocabulary ignore terms that have a document frequency strictly lower than 3. Further, the parameter, ngram\_range specifies the lower (here one grams) and upper boundary (here 2 grams) of the range of n-values for different n-grams to be extracted.

Also, I have used TfidfVectorizer i.e. CountVectorizer followed by TfidfTransformer. The TfidfTransformer is used to transform a count matrix to a normalized tf or tf-idf representation. Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency. This is a common term weighting scheme in information retrieval, that has also found good use in document classification. The goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus. For example, words like if, a, the, we etc. appear a lot in sentences and thus their frequency in a corpus will be really high. This effect is nullified by tf-idf transformer.

The formula that is used to compute the tf-idf of term t is tf-idf(d, t) = tf(t) \* idf(d, t), and the idf is computed as idf(d, t) = log [ n / df(d, t) ] + 1. If smooth\_idf=True (the default), the constant “1” is added to the numerator and denominator of the idf as if an extra document was seen containing every term in the collection exactly once, which prevents zero divisions: idf(d, t) = log [ (1 + n) / (1 + df(d, t)) ] + 1.Here is how I have used it:

*(TfidfTransformer(norm='l2', smooth\_idf=True, sublinear\_tf=False))*

norm='l2' in the parameter here, means that the normalization is cosine, for the TfidfTransformer. Moreover, Smooth idf weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once, it prevents zero divisions and we applied sublinear tf scaling, i.e. replace tf with 1 + log(tf).

**Details of Execution:**

To understand the overall approach, first we open the twitter-training-data.txt file and then we loop through the lines of the file stripping default whitespace characters from each of the lines and further Splitting the line on one or more tabs/spaces of file. The cleaned terms are appended inside the list “list”. To do further operation on the list the list is converted to a pandas dataframe and the columns inside the list are named to tweet\_id, sentiment, tweet\_text to represent the respective members of the tweets. Then the tweet\_text column of the dataframe is accessed and the contents are filtered by sending them as a parameter to the preprocessing function. Further, the stop words are removed from the preprocessed twitter training data and it is assigned to the tweet\_text. Finally, we join the words under the header token\_text with space and assigns it to dataframe header tweet\_text.

**Training of the classifiers**

For training purpose here, I have made use of classifiers logistic regression and Multinominal Naïve Bayes classifier.

In the first case of training of the logistic regression, it has been used with pipeline. We make use of the Pipeline of transforms with a final estimator. Sequentially apply a list of transforms and a final estimator. Intermediate steps of the pipeline must be ‘transforms’, that is, they must implement fit and transform methods. The final estimator only needs to implement fit. The transformers in the pipeline can be cached using memory argument.

In our case the initial transfomer is ('vectorizer', CountVectorizer(min\_df=3, ngram\_range=(1,2))) and the final estimator is logistic regression classifier, ('classifier',LogisticRegression(random\_state=0). Later, the pipeline is used to fit the model, with the training data as values form the column tweet\_text from the list dataframe and the values from the column sentiment form the list dataframe training data. The results are as follows:

precision recall f1-score support

neutral 0.57 0.34 0.43 363

positive 0.56 0.78 0.65 983

negative 0.73 0.56 0.64 1033

avg / total 0.64 0.62 0.61 2379

Details of the training of the **second classifier** are as follows:

At first an empty list development is defined. Then the lines of the file twitter-test3.txt are read in a loop and are stripped of white space characters and further split the lines on tab spaces and append the values in a list. Further, three different lists are defined which are testing, list\_t, identity. Then the development list is converted into pandas dataframe, removing the stop words form the tweet\_text and assigning it to token\_text. Predict using the linear regression model for the list testing that contains the preproceesed tweets text only with stop words removed, from file twitter-test3.txt. Accuracy score predicts the accuracy of the correct predictions. Here list\_t stores the true values of twitter sentiments and prediction holds the predicted values of twitter sentiments. Classification report shows the main classification metrics. It takes the lists list\_t, prediction as input and target names is set to the values of sentiments. The results are as follows:

precision recall f1-score support

neutral 0.23 0.73 0.35 114

negative 0.83 0.55 0.66 1480

positive 0.56 0.73 0.63 785

avg / total 0.71 0.62 0.64 2379

**Third Classifier:**

For the third classifier almost everything remains same as the second except the classifier here is MultinomialNB(). The results obtained are as follows:

precision recall f1-score support

positive 0.04 0.93 0.07 14

negative 0.84 0.50 0.63 1630

neutral 0.50 0.70 0.59 735

avg / total 0.73 0.57 0.61 2379

Among all the three models, the highest average f score is 0.64 for the 2nd classifier model, so I will prefer that logistic regression classifier.