ASSIGNMENT2:
CLASSIFIER FOR SENTIMENT ANALYSIS ON MOVIE REVIEWS

UNI nr2483

Language Chosen: Python

Preprocessing:
The contents of the review are first split into tokens by using the re.split function.
Then punctuations are removed using regular expressions. This is followed by removal of stop words and Stemming of the words before storing in the dictionaries. NLTK is used to get the standard list of stop words and Porter Stemmer is used for stemming

On performing the first test run with k=7000 and submitting results on Kaggle, certain additional preprocessing checks were added to improve the accuracy. I made following changes:

While removing stop words I have retained the words "not","no" and "nor" as these contain information. Also additional checks for '/' and "" which were not handled earlier were added.

Feature Extraction:
All the terms are extracted and stored in the following dictionaries:
freq_dic : stores the words with the count of the number of occurrences
pos_feature_dic : stores the words occurring in positive reviews and a total frequency count
neg_feature_dic : stores the words occurring in negative reviews and a total frequency count

Feature Selection:
I have used the Chi-squared statistic for feature selection which is a standard statistical test to measure the independence of two events(i.e how much of an impact the current word will have on the classification as positive or negative)
I have sorted the list of all features in descending order of the Chi-squared values in order to choose the best k features having higher Chi-squared values.

Experiment Results with different values of feature counts (Best 'k' features shortlisted for classification)

k	Accuracy
3000	74.77%
7000	64.43% (due to inaccurate punctuation removal)
8000	74.33%
10000	75.49%
16000	76.47% (without stemming)

The highest accuracy was observed for 16000 (which is almost the entire vocabulary) However this unexpected result could be because this is a relatively small data set.
However overall based on the other tests it was observed that overall k=3000 gives good results of 74.77% which is the next best result. Since there are only 16014 features in all, I have selected k=3000 as a good measure to build the classifier as its the best tradeoff between the ratio of selected features to the total features and the accuracy.

Classification Algorithm: Naive Baye's
Why Naive Baye's Classifier?
Even though it makes an oversimplified assumption that all the words are completely independent,Naive Baye's has proved to be an effective classifier as it is based on maximum likelihood Naive Baye's is easier to train and works well with small data sets. It requires only a small number of parameters to train in comparison to some of the more complex classifiers and provides good results.

Design decisions/ Approach:
To calculate the Chi-squared test for feature selection, the following 4 stats were needed for each feature:
1) Number of reviews in which the feature was present and were positively classified f11
2) Number of reviews in which the feature was present and were negatively classified f10
3) Number of reviews in which the feature was not present and were positively classified (can be calculated as pos_review_count-f11)
4) Number of reviews in which the feature was not present and were negatively classified

Now the chi-squared value can be calculated using the following formula:

```
chisquared_dic[feature]=((float) (((f11+f10+f01+f00)*pow((f11*f00-f10*f01),2))/(float)((f11+f01)*(f11+f10)*(f10+f00*f01+f00))))
```

To calculate the probability score, we need the following :
prior probability of a positive review (calculated by using a counter pos_review_count while reading training data,training_example_count to keep track of the total reviews)
prior probability of a negative review (calculated by using a counter neg_review_count ,training_example_count to keep track of the total reviews while reading training data)

To calculate the probability scores, we need a product of the probabilities of all the tokens which are very small (some even as in the order 10^-5) So to avoid a floating point underflow errors logs of the probabilities have been used which doesn't affect the comparison in any other way. Also there could be terms which are seen for the first time which would result in a probability of zero and log (zero) is not defined. So we use smoothing by adding one to the numerator for all the tokens

For feature selection I needed a feature list in descending order of Chi-Squared values. To sort over a hash , the sorted method of dictionary is used as follows:

```
sorted_chisquared_dic=sorted(chisquared_dic.iteritems(),key=operator.itemgetter(1),reverse=True)
```

Wherever possible the code has been optimized. For example instead of using a for loop to compute the sum of probabilities, I have used built-in functions of Python lists like 'reduce' as follows:

```
pos_prob_total = reduce(lambda a,d: a + d,token_pos_prob_list)
```

Based on the value of k (which can be set in the code), the first k elements are selected from sorted_chisquared_dic which contains the features in descending order of their chi-squared values. The

remaining features are deleted from the training model.

Features:
1) The trained models are stored as pickled indexes to avoid training everytime testing needs to be done. Training module may be called only when new documents are added or the value of k needs to be changed or there is some improvement made in pre-processing
2) The code is optimized using as many built-in functions as possible of Python such as “sorted” and “reduce” instead of using for loops to compute these to minimize the time complexity
3) Various statistics such as the time needed to train model, test data, memory usage are printed in the Console
4) Hard coded paths are not used for training and testing data files and is taken as input from the user

References:
Introduction to Information Retrieval by Christopher D Manning
http://en.wikipedia.org/wiki/Naive_Bayes_classifier
http://familypedia.wikia.com/wiki/Naive_Bayes_classifier