Sentiment Classification

Input

The program uses 'reviews.txt' file as input. Reviews from IMDB, Yelp and Amazon are combined to form the reviews.txt hence it contains all the 3000 reviews.

First, we store the reviews and corresponding sentiment labels into two different lists, reviews and label respectively. All the steps of pre-processing are carried on this list with textual data.

Classifiers

The following classifiers are implemented:

1) Multinomial Naïve Bayes Classifier
2) Support Vector Machine

Implementation Details

1) Multinomial Naïve Bayes Classifier: This classifier has been implemented without using standard packages such as Sci-Kit. The feature here would the word count. Therefore, the classification of a sentence would be based on the number of times the word occurs in a sentence.

   Assumptions:
   i) The order of the words doesn't matter and a sentence in characterized by just the words occurring in it.
   ii) Relative word frequency is treated as a feature.
   iii) Each feature is independent from others (Naïve Bayes assumption).

   Simplified representation:

   $P(c \mid d) = \max \{ P(c) \prod_{i=1}^{n} (wi \mid c) \}$ for all c

   $P(c_i)$ = Number of documents of class i / total number of documents

   $P(w_i \mid c)$ = Number of times w_i appears in class c / Total num of words in class c

   Implementational changes:

   i) To avoid floating point underflow, we take log on both sides and further calculations are carried out.
   ii) Laplace smoothing is used to check cases of zero probability.

   Data pre-processing: The above discussed method is implemented after pre-processing the given data.

   i) Numbers and special characters are stripped of the given data
   ii) The documents are tokenized
   iii) Stop words are used to eliminate words that commonly occur but doesn't add value as a feature.

Data Split: Since validation set doesn't play any role in Naïve Bayes Classifier, that is split into Train and Test sets only. 80% of the total documents are used for training and the remaining is used to testing and accuracy prediction.

Feature Selection: The implementation is based only on the count of occurrences of words. However, for comparison NB classifier from Sci-Kit package is implemented with Tf-Idf vector as feature.

Offline Efficiency Cost: The time taken to build the classifier is around 7 seconds.

Time to classify a new tuple: Approximately 0.0003 seconds.

Accuracy:

	Base Implementation	Implementation using Sci-Kit package
Count Vector	83.5%	81.66%
Tf-Idf	NA	80%

Comments:

i) The above table clearly shows the difference in the accuracies for different features used. For comparison, during the implementation using Sci-Kit package, the stop words list wasn't modified i.e. words such as 'NOT', 'NOR' which might play a key role in polarity of a sentence aren't removed from the stop-words list. Hence the prediction accuracy is lower than that implemented from scratch.

ii) Stemming and Lemmatization: Though stemming and lemmatization was used in the sci-kit package implementation, it doesn't have much difference on the prediction accuracy. This might be because of lack of various different forms of words used in the given data.

iii) Feature Selection: Tf-Idf implementation of MNB classifier surprisingly yields lower accuracy. Intuitively, this might be due to the smaller size of the dataset and hence lower frequency of words.

2) Support Vector Machine: Sci-Kit learn package is used to implement this classifier.

Data Pre-processing: The pre-processing is similar to the one described in the NB classifier.

i) Numbers and special characters are stripped off the document and all words are in lower case.

ii) The documents are tokenized.

iii) Words are stemmed and lemmatised to get root word.

Feature Extraction: Count Vector and Tf-Idf features are usedt for classification.

i) Number of the words: Count of the words is used as feature i.e. each unique word in the vocabulary list is a feature. Count vectorizer typically gives more weightage to longer sentences as the number of words is more.

ii) Tf-Idf : To overcome the drawback of CoutVectorizer, we use Tf-Idf which intuitively represents the importance of each word in a document or dataset in general.

Data Spilt: We use grid search to find the best parameters using the validation set. Initially data in split as 85% train data and 15% test data. To create validation set, the training set is divided into 75-25% split, 25 being the validation data.

Offline Efficiency Cost: Due to the validation procedure i.e. to find the parameters that give the best results the time taken to build the classifier is high. It takes more than 2 hours(on a machine with 8gb memory) to find the best fitting parameters on the first run. Hence, this part of the code is commented as we already have the best fit and the parameters can be used to fit the model.

Parameters **C** (used to optimize the selection of hyperplane), **kernel** (we test for Linear and rbf kernels), **Gamma** (kernel parameter for rbf), **Probability** (to enable probability estimates while fitting the model) are used.

The values obtained for these parameters are used further.

Time taken to build the model after validation: Around 5 seconds

Time to classify a new tuple: ~0.0006 secs

Accuracy:

	Count Vectorizer	Tf-Idf Vectorizer
Test	83.5%	82.4%
Training	97%	89%

Comments: The validation step helps us get the parameters in order to get the highest classification accuracy on test data. Though a complex model is used, the results are not better than that of NB classifier. Possible reasons could be the size of the dataset.

The pruning helps in reducing the running time.