CSE508: Information Retrieval - Assignment 4

- *Shreyash Arya (2015097)*

For this assignment, Navie Bayes is implemented in the same fashion as mentioned in the class lecture slides and add 1 smoothing is used to tackle the unknown words.

Pre-processing is done similar to previous assignments: The following steps are taken (in the mentioned sequence):

- Removing HTML tags using BeautifulSoup inbuilt module.
- Fixing the contractions (Eg. don't → do not) using contractions python library.
- Tokenization using NLTK's word tokenizer.
- Removing non-ascii characters using Unicode.
- Lowercasing the document data.
- Removing punctuations using regex (r'([^\w\s])|_+').
- Replacing numbers with words using the num2words library.
- Stemming using NLTK's Porter Stemmer.
- Removing stopwords using the NLTK's English stop-words dictionary.

Data splitting is done by providing the ratio parameter and division is done with equal balance in each class.
Different dictionaries are built on the go for the efficient run of the program.

Naive Bayes:

Naive training is done on the train data that is provided from the data splitting function. Priors and conditional probabilities are calculated for the train set with add 1 smoothing kept in mind.

[Prior for each class k
*= np.log ((# of docs in class k * 1.0) / (# of doc in train data))]*

[Conditional Probability for each term t for each class k
*= np.log (((# of occurences of t in class k * 1.0) + 1) / (total # of terms in class k + |Vocab|))]*

At the time of testing, the test data is fed into the trained classifier which calculates the maximum a posteriori class or the class which is the most likely to which the document belongs. If the term is not present in the training set for which the conditional probability is not present, it has given a score of log(1/|Vocab|)*100 which is penalized 100 times.

[Score = argmax over all classes (Prior of class + Sum of the conditional probability of all terms in the document)]

TF-IDF Feature Selection:

Top k (k is taken as the 60% of the total data fed into the classifier) features based on the TF-IDF values are used to select the terms that will contribute to the score of a class i.e. the conditional probabilities of the terms which are present in the top k list are only added to the score and others are treated as unknown words and given score as mentioned above. Single TF-IDF score is considered for a term which is cumulative over all the documents in the train dataset.

Results and Inferences:

```
Ratio: 0.5
Accuracy: 95.2%
True vs Predicted
[[493.   1.   0.   3.   3.]
 [  0. 482.   7.   5.   6.]
 [  1.  10. 477.   7.   5.]
 [  5.  24.  25. 439.   7.]
 [  1.   5.   5.   0. 489.]]
```

```
Ratio: 0.7
Accuracy: 95.7333333333%
True vs Predicted
[[297.   0.   0.   1.   2.]
 [  0. 284.   4.   9.   3.]
 [  1.   4. 288.   6.   1.]
 [  1.  10.  11. 277.   1.]
 [  1.   2.   7.   0. 290.]]
```

```
Ratio: 0.8
Accuracy: 96.1%
True vs Predicted
[[200.   0.   0.   0.   0.]
 [  0. 187.   4.   7.   2.]
 [  1.   1. 194.   4.   0.]
 [  0.   6.   6. 187.   1.]
 [  1.   2.   4.   0. 193.]]
```

```
Ratio: 0.9
Accuracy: 98.0%
True vs Predicted
[[100.   0.   0.   0.   0.]
 [  0.  95.   1.   4.   0.]
 [  0.   0.  99.   1.   0.]
 [  0.   1.   2.  97.   0.]
 [  0.   0.   0.   1.  99.]]
```

We can see from the results that as the train data increases, the test accuracy also increases ranging from 95% to 98%. There can be a possibility of overfitting when the 90:10 ratio is considered as we get a very high accuracy.

In the second part, if we choose the features carefully using the TF-IDF scoring, then weight to only those terms should be given which are important to the class.

Ratio: 0.7
Accuracy: 96.2%
True vs Predicted

```
[[295.   0.   0.   1.   4.]
 [  0. 282.   5.   7.   6.]
 [  1.   5. 285.   6.   3.]
 [  0.   2.  10. 287.   1.]
 [  1.   2.   3.   0. 294.]]
```

The accuracy for 70:30 ratio is increased as compared to part 1 by 0.5% approx which shows that better feature selection and removing noisy features helps in better prediction.