Building a App Review Classifier

1. Approach

This project is done in Python mostly using the Natural Language Toolkit (NLTK) package. The entire experiment can be categorized into 4 steps: 1) Data Pre-processing and Sampling, 2) Selection of Features, 3) Training, Testing and Evaluation 4) Analysis

2. Data Pre-processing and Sampling

Firstly, I wrote a script to load all the data from all the different .csv files into a list of tuples containing (review, genre). While parsing the reviews, I converted the sentences to lowercase, removed non ASCII values and punctuation. I then tokenized the sentence and removed stop words provided by the NLTK package. I also skipped single character tokens and reviews that were less than five tokens long after removing stop words. I feel that doing this would provide that classifier with a good set of data to train with. Lastly, I saved the list using the Pickle library. The Pickle library allows the storing of Python objects as a file.

I then wrote a different script to perform quota sampling on the list of tuples. This ensures that each genre of apps has the same number of reviews to train with. I did a quick survey of the distribution of reviews amongst the genres.

The weather only had 60,000 reviews after the filtering while the others had more. I decided to randomly sample 10,000 from each genre, amounting to a final training data set of 50,000 reviews. The random sampling was done by shuffling the data and taking the first 10,000 of each genre. While storing the sampled data, I also complied all the words contained in the reviews. This is used at the next stage, selection of features. Similarly, I separately sampled 2,000 reviews for each genre for testing purposes giving a testing data set of 10,000 reviews.

Stemming was not done as I felt that information would be lost at the expense of just having less features. The problem of too many features can be easily mitigated by carefully selecting the features.

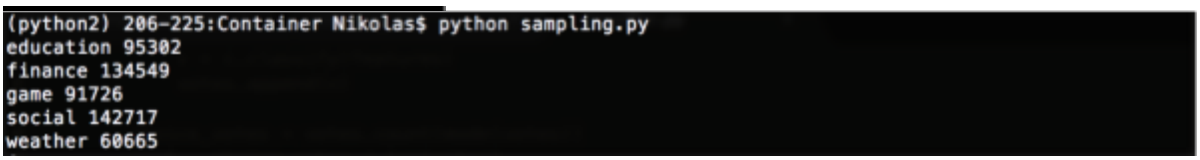
3. Selection of Features

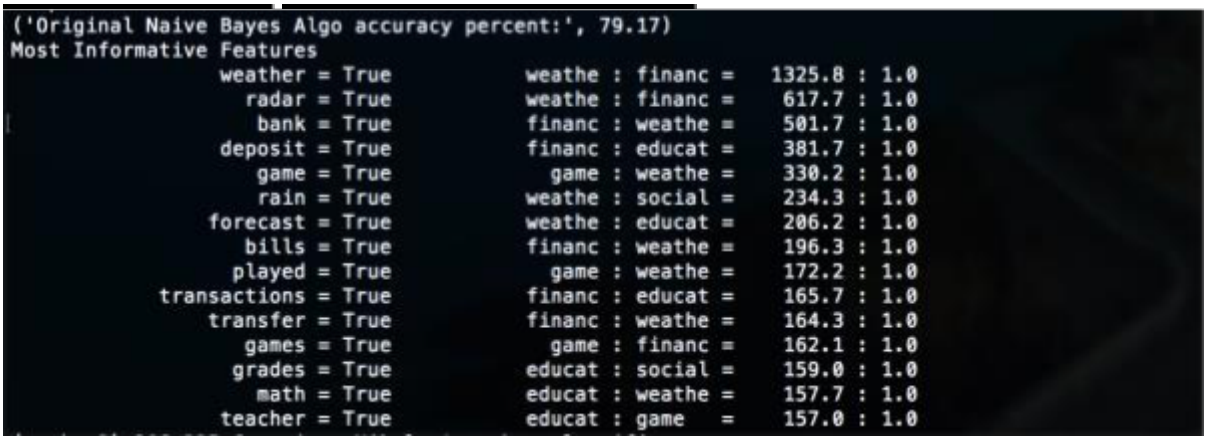
I first surveyed the distribution of words contained in all the reviews. This was done using the freqdist method which returned a dictionary of all the words and their word counts. I used this to view the top 50 words across the genres. I felt the most of these words were meaningful, perhaps proof of the effectiveness of the earlier pre-processing methods. I then took the top 5000 words and used them as features.

4. **Training, Testing and Evaluation**

I first converted the data, into a list of feature dictionary and genres. Each feature dictionary has a Boolean value for each feature. I then split the data into 200,000 for training and 50,000 for testing. I used the Naïve Bayes Classifier provided by the NLTK package as it is supposedly quicker than other learning algorithms. However, this still proved to take quite a long time.

The classifier got an accuracy score of 79.17%. I also took a look at the 15 most meaningful features determined by the classifier using an in built method:





Most of the features identified were actually meaningful and intuitive, confirming the validity of the classifier. For example, words like “rain”, “weather” and “forecast” are obviously related to the weather while a word like “radar” requires investigation.

5. Analysis

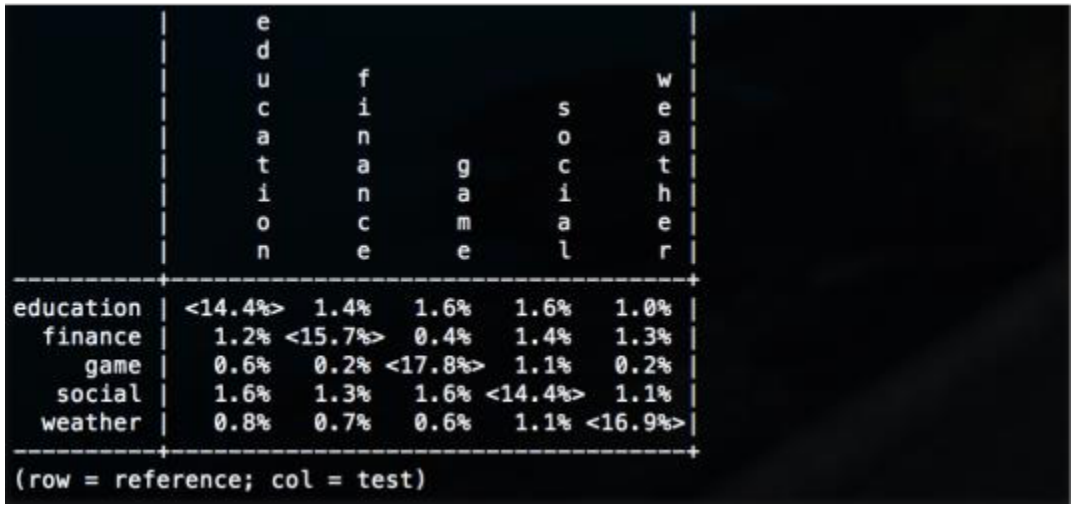
Despite the decent accuracy score, this model is likely to fail at classifying shorter reviews with less meaningful words due to the way that it was trained on “higher quality” data. However, this is acceptable as most classifiers are likely to struggle with shorter reviews as well. The classifier is also likely to struggle if it includes more obscure words that were not selected as part of the 5,000 features in section 3. This is also acceptable as the users are unlikely to completely avoid the 5,000 features and having some should be good enough information for the classifier.

Moving forward, there are some methods to further improve the classifier's performance. One way is implementing more algorithms and take a majority vote with a confidence level (most votes / total votes). The classifier can then choose to make predictions for reviews that produce a high confidence value. Depending on the use case, the classifier can "admit" that it is unsure and just not classify some reviews, resulting in a possibly higher accuracy for the reviews that it chooses to classify.

I have written the code for such algorithms but did not implement them as I did not have the computing power to train that many classifiers.

Lastly, to do further analysis on the errors made by the classifier, the ConfusionMatrix method is used.

From the confusion matrix, it appears that the errors are more or less evenly distributed amongst the comparisons. Hence, little can be done to resolve classification errors between any specific two genres.



	education	finance	game	social	weather
education	<14.4%>	1.4%	1.6%	1.6%	1.0%
finance	1.2%	<15.7%>	0.4%	1.4%	1.3%
game	0.6%	0.2%	<17.8%>	1.1%	0.2%
social	1.6%	1.3%	1.6%	<14.4%>	1.1%
weather	0.8%	0.7%	0.6%	1.1%	<16.9%>

(row = reference; col = test)

6. Conclusion

In conclusion, based on the experimental results, I believe that it is viable to perform the task given with the datasets provided with the above implementation. However, the classifier produced would only work well on "higher quality" reviews which consist of enough features to be picked up. In cases where that is not the case, the problem can be mitigated by the confidence threshold implementation in section 5 to at least highlight classification choices made with low confidence.

References:

NLTK - http://www.nltk.org/

Pickle - https://docs.python.org/2/library/pickle.html

Naïve Bayes - http://www.nltk.org/_modules/nltk/classify/naivebayes.html

Confusion Matrix - http://www.nltk.org/_modules/nltk/metrics/confusionmatrix.html