YELP SENTIMENT ANALYSIS

EECS 4412
Barrett Tran
214395735

INTRODUCTION

Text classification is a form of natural language processing that relies on supervised machine learning. Sentiment analysis is a subcategory of natural language processing that is concerned with extracting the polarity of text. The objective of this project is to build a classifier that categorizes Yelp reviews into one of three classes: *positive, negative, neutral*. With 40,000 training examples, a classifier will be built to predict the class of 10,000 reviews. The goal is to maximize classification accuracy when analyzing reviews.

PREPROCESSING

Stopword removal and stemming was done using a python script that takes as input the train and test csv and outputs the train and test arff files. Non-alphanumeric characters were also removed. Since stopwords do not aid in classifying the sentiment of a review, they are removed which can slightly improve model training and testing performance. The stemming algorithm used was part of the natural language processing toolkit (NLTK). I used SnowballStemmer over PorterStemmer because Snowball is the second iteration of Porter with better stemming techniques. I stemmed the words because since training time with the Random Forest algorithm is relatively long, reducing words to their root would reduce the number of features after attribute selection and it could also leave space for other potentially useful classifying words.

I begin by adding a few stopwords to the text file. Specifically some stopwords/contractions that do not contain an apostrophe. For example, ‘cant’, ‘theyve’, ‘youll’.

The training data contains 40,000 examples. The class distribution is exactly 22,341 positive, 10806 negative, and 6853 neutral reviews. To normalize this class imbalance, the resampling filter in WEKA was used with a bias of 1 and the resulting data contained 13,333 examples in each class. Since the class distribution of the test data is unknown, UI

The StringToWordVector filter was used with wordsToKeep parameter set to 2000 to capture most of the common review words. I then used AttributeSelection with GainRatio as the evaluator and Ranker as the search method with a threshold value of 0. The resulting data contained 1733 attributes/word vectors.

CLASSIFYING ALGORITHM

I chose Random Forest to train my model. Random Forest had the highest classification accuracy from my testing with training splits and 10-fold cross validation. This may be due to bootstrapping and/or bagging because I found that increasing the number of trees gave more accurate results but at the cost of training time which was very long compared to an algorithm like MultiNaiveBayes. Max depth was left at unlimited to prevent underfitting. The number of attributes had a direct effect on training time as doubling the number of features almost doubled the time to train the model. For this reason, I used GainRatio to select the top 2000 words and split the training data at 66% to test the model.

RESULTS AND CONCLUSION

Model was trained on a training split of 66%. Ideally, 10-fold cross validation would be used but training time was not ideal.

The difference in classification with and without stopwords was negligible at < 1% so I opted to remove them to potentially increase the amount of words that add sentiment to a review. Similarly, the difference between stemming and not stemming was negligible but I opted to stem for reasons previously mentioned.

After all preprocessing was done, I adjusted the number of trees parameter to try and optimize the classification accuracy. 100 trees resulted in a classification accuracy of 86.9% while < 100 had a decreased accuracy. For the sake of efficiency, I did not exceed 100 trees.

A possible improvement in classifying reviews may be to spellcheck incorrectly spelled words using a library and script. Maximizing the number of trees and words to keep to train the model would also improve classification but at the cost of being inefficient.

APPENDIX

Included programs:

- clean.py

To run clean.py, use command “python clean.py” with stopwords.txt, train.csv, and test.csv in the same directory. The program will output train.arff and test.arff files.

I ran weka with an 8GB heap to stop memory crashes.

- “weka.jar -Xmx8g”