INFO 290 Project Report: Predicting Star Ratings from Reviews

Edward Kuanshi Lu

I. INTRODUCTION

For my final project, I tried to predict the number of stars a user would give, given that user's review. I split the reviews into n-gram hashes and used these as feature vectors for a variety of classifiers. I compared the classifiers and found that the Naïve Bayes classifier performed the best.

II. PROBLEM

The dataset I'm using is the Yelp Academic Dataset. [1] This dataset has JSON objects full of Yelp reviews with their associated star ratings and is perfect for my purposes.

The immediate motivation for my problem is that when a user types a review, my software could suggest how many stars the user should give. A suggestion would be better than the user choosing himself/herself because sometimes users give biased star ratings (e.g. they just tend to give 5 stars because that's what they always do). A predictor which learned through a large sample of users' reviews would be able to provide a more objective star rating, which may lead to better information.

A more general, and perhaps more interesting application of a solution to this problem would be in the field of natural language processing (NLP) and speech recognition. If I can figure out the "star rating" of some paragraph (spoken or otherwise), then I effectively determine some part of the tone the author wanted to get across. For example, if the writer writes a particularly positive paragraph, I would predict it would have a star rating of 5 stars. This could be helpful for NLP because if it encounters words that may have double meanings, it would choose the one that is more positive. It would help guide speech recognition search by limiting the search space to more positive words.

A stretch (but very interesting) application of this work may be possible in the form of a sarcasm detector. If a piece of writing tended to be very negative (1 star) and then suddenly has a very positive (5 stars) sentence, that sentence is likely sarcastic. This would also sometimes work for the vice versa case.

III. SOLUTION

I first processed the dataset to produce a few sub-datasets. I didn't want to use the entire dataset because I would run into memory problems with my classifiers. Instead, I created 2 sub-datasets with 100 reviews per star (1 for training, another for testing), another sub-dataset with 1000 reviews per star, and another one with 10,000 reviews per star. I also hypothesized that limiting these reviews to one category would improve classification, so I created another 3 sub-datasets like the ones before, but restricted to reviews of restaurants (i.e. the JSON object's business_id matched a business whose JSON object had a category of "Restaurants").

I then produced 3 python pickle objects of labels (the star rating) and features (the n-gram hash) from each sub-dataset. The first pickle object would contain only unigram hashes, the second would contain bigram and unigram hashes, and the third would contain trigram, bigram, and unigram hashes. I wanted to do this because I hypothesized that some bigrams and trigrams may be very useful for classification (such as the phrase "not bad" and "not very good"). From here on, we refer to these pickle objects as <num reviews> <n-gram> <(restricted)> datasets. For instance, the 10,000 unigram dataset is the non-restricted, unigram dataset of 10,000 reviews.

I used an excerpt of the dataset, so it was not too large (less than a gigabyte). Therefore I did not use map reduce to process it, but instead opted for sequential processing.

I will now discuss the results and problems of each classifier I tried. The results are summarized in Figures 1, 4.

[1] https://www.yelp.com/academic_dataset

I started the learning process using a SVM classifier. Not only was it incredibly slow to train, but the results were not promising. My highest accuracy rate was from the 10,000 unigram sub-dataset, producing an accuracy rate of 28%. This dataset was not restricted to restaurants, which was surprising because the one that was restricted to restaurants produced a lower accuracy rate of 27.2%. Adding bigrams and trigrams did not help the accuracy rate either. For example, the 10,000 bigram dataset produced 20.15% accuracy and the 1,000 trigram dataset produced 21.15% accuracy. I could not learn over the 10,000 trigram dataset, because the SVM would run out of memory. However, it was not projected to do better than the 10,000 unigram dataset, because the 1,000 unigram dataset had an accuracy rate (23.6%) higher than the 1,000 trigram dataset. These accuracy rates were disappointing, because they are only slightly better than randomly guessing (which would on average produce a rate of 20%).

One of the strange aspects of this SVM was that it would tend to guess 5 stars. For instance, out of 500 (100 per star) guesses, it would guess 384 5 stars for the 10,000 unigram dataset. Because of this behavior, I did not use cross fold validation since this does not indicate an overfitting problem.

I next tried using a Naïve Bayes classifier. This produced much better results. The best accuracy rate I procured with this method was 53.2% from the 10,000 unigram restricted dataset. Strangely enough, restricting the dataset to Restaurants worked better for Naïve Bayes than for SVM. The 10,000 unigram dataset produced 52%.

Next I tried using χ^2 scores to rank the most important features (the ones that split the classes the best). I then trained my Naïve Bayes classifier using only those features. It ran much quicker (minutes versus hours for SVM), but I tended to lose 0.5% from my accuracy rates.

I trained using a Decision Tree classifier next. However, it was not very memory efficient, so I could only train it on the 100 unigram dataset. It produced a 28.4% accuracy rate, which was better than SVM, but worse than Naïve Bayes on the same dataset.

Next I tried using a Random Forest classifier. It faced the same memory problems as the Decision Tree classifier, but produced a 29% accuracy rate, 0.6% better.

To alleviate the memory problems, I tried training the Decision Tree classifier using only the 1000 most important features. However, this did not help accuracy rates at all. For instance, training it on the 100 unigram dataset produced a 17.8% accuracy rate, worse than random guessing.

Finally, I tried combining the Random Forest, Naïve Bayes, and SVM classifiers. I didn't include the Decision Tree because I wanted an odd number of classifiers and Decision Tree classifiers are just a subset of Random Forest classifiers. I had them vote for the correct label and return the label with the majority votes. If no one agreed, I would return Naïve Bayes's result. I tried it on the dataset these classifiers best performed on (the 10,000 unigram dataset) and produced an accuracy rate of 45.2%, worse than Naïve Bayes on its own.

N-gram	Size	Restricted?	Accuracy
1	100	no	0.238
1	1000	no	0.236
1	10000	no	0.28
2	100	no	0.23
2	1000	no	0.206
2	10000	no	0.2015
3	100	no	0.2075
3	1000	no	0.2116
1	100	yes	0.228
1	1000	yes	0.228
1	10000	yes	0.272

Fig. 1: SVM experiments

N-gram	Size	Restricted?	With χ^2?	Accuracy
1	100	no	no	0.362
1	100	no	yes	0.346
1	100	yes	no	0.39
1	1000	no	no	0.45
1	1000	no	yes	0.478
1	1000	yes	no	0.47
1	10000	no	no	0.52
1	10000	no	yes	0.514
1	10000	yes	no	0.532
2	100	no	no	0.36
2	100	no	yes	0.35
2	100	yes	no	0.382
2	1000	no	no	0.432
2	1000	no	yes	0.474
2	1000	yes	no	0.496
3	100	no	no	0.374
3	100	no	yes	0.34
3	100	yes	no	0.348
3	1000	no	no	0.422
3	1000	no	yes	0.432
3	1000	yes	no	0.462

Fig. 2: Naïve Bayes experiments

N-gram	Size	With χ^2?	Forest?	Accuracy
1	100	no	no	0.284
1	100	no	yes	0.29
1	100	yes	no	0.178
1	1000	yes	no	0.224
1	10000	yes	no	0.208

Fig. 3: Tree and Forest experiments

N-gram	Size	Accuracy
1	10000	0.452

Fig. 4: SVM, Naïve Bayes, Tree combined experiment

IV. RELATED WORK

I used libsvm[2] for Python for my SVM classifier. I used their svm_train, svm_save_model, svm_load_model, and svm_predict functions.

I used scikit[3], nltk[4], scipy[5], numpy[6] for Python for my Naïve Bayes, Decision Tree, and Random Forest

[2]http://www.csie.ntu.edu.tw/∼cjlin/libsvm/
[3]http://scikit-learn.org/stable/
[4]http://nltk.org/
[5]http://www.scipy.org/
[6]http://www.numpy.org/

classifiers. I used their classifiers: RandomForestClassifier, MultinomialNB, and DecisionTreeClassifier. I also used their Pipeline, SelectKBest, and chi2 functions to rank the importance of features.

V. FUTURE WORK

Some further work could be done in this area. For example, I believe we may achieve better accuracies of we could incorporate the voting histories of that user. This, however, would restrict the solution and make it less useful for NLP type applications, because we'd need to know person who is speaking/writing.

Another thing I could do is integrate a spellchecker in order to normalize the reviews.