# Yelp Review Usefulness Classification and Prediction

Christina Ionides

Spring 2013

## 1 Summary

This paper discusses the process, techniques, and results included in classifying Yelp reviews as either “useful” or “not useful”, and the subsequent prediction of a review’s “usefulness” on a test set. The report has been divided into the following sections: Data, which describes the original data sources and format, Processing and Feature Selection, Classification, Evaluation, Prediction, and Conclusion. The “project” directory contains a README file that outlines the directory contents as well as an outline of all distribution folders. A list of resources I utilized while working on this project can be found at the end of the report.

## 2 Data

The data was obtained from the Kaggle Yelp Recruiting Competition, and can be viewed at: http://www.kaggle.com/c/yelp-recruiting/data. The data included a training and test set. In the training set there were 11,537 businesses, 8,282 checkin sets, 43,873 users, and 229,907 reviews. Of this data I only used the businesses and reviews. The data set was formatted so that each entry was a JSON object. Each business object had a type, business id, a list of neighborhoods, city, state, latitude, longitude, stars, review count, and a list of categories. Each review object had a type, business id, user id, stars, text, date, and a list of vote types and their counts. The votes could be either “’cool”, “useful”, or “funny”. Of the data provided for the businesses, I used the id, stars, and category list. For the review data I extracted the business id, user id, stars, text, and useful votes.

After processing the data there were 87,359 reviews from 5,779 businesses. The process is outlined below.

## 3 Processing and Feature Selection

I decided to narrow the reviews to one domain, restaurants and food. In order to better train the classifier, I chose the top and bottom third of each businesses review set. This reduced the training data to *87,359 reviews. In order to classify the reviews I built a feature table for each review instance. In selecting the features I chose a combination of structural, lexical, and meta-data features. The length of non-stopword tokens in a review, noun phrases, and the star rating comprised the feature set for each review. Each review was labeled as either “useful” or “not useful” in the table. To summarize, each row represented a review and each column represented a feature.

In order to get the data in this format, a good deal of pre-processing was needed. The processing can be broken down into four parts. The reading in and parsing of the Yelp training data, tagging the text with POS tags, extracting the features, and building the review feature table.

Part one of the processing was done by a Java application that I wrote, ReviewProcessor.java. The user, business, and review JSON objects are read into memory, although I only used the business and review data. The application uses a JSON parsing library, gson. Upon being read-in the data is parsed and Business and Review objects are created, then added to their respective maps. As mentioned in the previous section, for the review data I extracted the review id, the business id, star rating, and the usefulness rating. For the business data I extracted the business id, categories, star rating, and review count. While reading in the business data a list was maintained of all business ids that contained the categories “Restaurant” or “Food” in their list of categories. Then when reading in the review data, an object was only created if the business id was in the list of business ids that corresponded to restaurants or food. The review and business object are written to their own text files. When the review and business objects are being created and stored each business’ review set is being tracked as well as the total useful vote count for each review. This allows the application to select the top and bottom third of each businesses review set based on their useful vote count. This leaves a total of 87,359 reviews where the text is pulled and printed out separately for tagging. The application prints out 500 text files of equal size to make it easier for the POS tagger.

The second part of the processing is the easiest, but the most time-consuming. It involves tagging all of the text files from the previous phase, with POS tags. This is done in order to later extract noun phrases and count their frequencies. To do this task I used the Stanford Core NLP processor. I passed each text file in and an xml file of the tagged text was generated.

The third part of the processing was done by an XML parser that I wrote, XMLParser.java. It was used to read in all of the XML formatted and tagged reviews. The parser, first extracts noun phrases from the tagged text and prints them to a file, that is immediately read back in. The noun phrases are now cleaned up by removing white spaces, removing stop words, stripping punctuation, and making all words lower case. When all of the words have been “cleaned” they are added to a map, along with a frequency map. The final output of the parser is a noun phrase frequency map for all of the Yelp reviews. It is from this list that I selected the top 25 noun phrases as features for the feature table.

The fourth and final step of the processing is done by another Java application, FeatureLoader.java. As previously mentioned, the feature table consists of a review instance’s star rating, review length of non-stop word tokens, and a bit vector representing the 25 noun phrases selected in the previous phase. In order to label the review as useful or not, the useful vote count is considered. If a review has a useful vote count greater than or equal to one it is labeled with a “1” for useful and a “-1” otherwise. A hash map was employed once again to store each review entry and its feature list as the reviews were iterated over. The final output was printed to “review_features.txt”.

## 4 Classification

I classified the data using a linear SVM with a five-fold cross validation. The cross validation was performed using RapidMiner. As a parameter for the cross-validation I chose stratified sampling for when it would partition the data into training and testing sets. With 27 attributes and a label, the SVM cross-validation took approximately 4 hours to perform over 87,359 reviews. The program output the performance vector, a model, and training data. The results of the performance can be seen in Evaluation. TODO: After the initial run, I removed the attributes with the lowest weights to see if it would improve performance.

## 5 Evaluation

**Note:** The following results are from a smaller set of reviews(68,000). This is due to a bug I found in my code that had left out some reviews. I fixed the bug, and as you will see in all other documentation and results the larger set is discussed. Ideally, this would be run over the larger set of 87,359 reviews.

The performance vector generated by RapidMiner showed that the precision was higher when predicting useful reviews while the recall was higher for not useful reviews. Overall, the accuracy was 65.61%. The recall and precision for both the “useful” and “not useful” classes can be seen in Table1. Surprisingly, the review length had the worst performance of all the features, with a weight of -1.265. The top performing features were the star rating and the noun phrase “staff”’. The top ten features and their corresponding weights can be seen in Table 2 below. Additionaly, there are screen shots of the work process in RapidMiner_Classification. Total number of Support Vectors: 68008 Bias (offset): -0.241

| Class | Accuracy | Precision | Recall |
|---|---|---|---|
| Useful | 65.61% +/- 0.14% | 69.68% | 61.37% |
| Not Useful | 65.61% +/- 0.14% | 62.09% | 70.33% |

Table 1: Performance

| Feature | Stars | food | service | restaurant | order | server | bit | dinner | special | staff |
|---|---|---|---|---|---|---|---|---|---|---|
| Weight | 0.073 | 0.006 | 0.079 | 0.029 | 0.033 | 0.049 | 0.054 | 0.03 | 0.028 | 0.053 |

Table 2: Top Feature Weights

## 6 Prediction

WIth the model that was generated from the classification process I was able to make "usefulness" predictions on a test set of Yelp reviews. There were 22,956 reviews in the Yelp test set. I wrote a FeaturesLoader program almost identical to the one mentioned previously differing in that there was no useful vote count for this review set. The same filtering of restaurant reviews was done,and the same set of noun phrases were read in to build the feature table with. The resulting test set had 6,647 examples in it. There were far more "not useful" predictions, 2458, then "useful", 4189. The following statistics were generated by RapidMiner.

| Role | Type | Statistics | Range |
|---|---|---|---|
| confidence useful | real | avg = 0.482 +/- 0.196 | [0.183 ; 1.000] |
| confidence not useful | real | avg = 0.518 +/- 0.196 | [0.000 ; 0.817] |
| prediction | binomial | mode = -1 (4189), least = 1 (2458) | 1 (2458), -1 (4189) |

Table 3: Prediction Statistics

## 7 Conclusion

This approach of classifying reviews could have several use cases. Since Yelp and the user already know if a review is useful by its useful vote count, why bother classifying? It can lead to further insights on what comprises a useful or not useful review. This information could be used to help someone write a better review. For instance, if a user is writing a review on a restaurant and they have not included anything about the service, yelp could suggest including this in the review. A more obvious use case is that of classifying reviews that have no votes yet. It is typical for a review that is well liked to keep getting viewed and voted on. With predicting a review's usefulness Yelp can recommend a wider variety of reviews than it would have previously.

It would be interesting to apply this kind of process across multiple domains to see what kinds of features make a good review regardless of the topic. With more time I would have liked to compare the SVM results with those of other classifiers. More importantly, I would like to refine the feature table to optimize performance, perhaps by adding different types of features in.

## References

[1] Michael P. O'Mahony and Barry Smyth. Learning to recommend helpful hotel reviews. In *Proceedings of the third ACM conference on Recommender systems* (RecSys '09). ACM, New York, NY, USA, 305-308.

[2] Minqing Hu and Bing Liu. 2004. Mining opinion features in customer reviews. In *Proceedings of the 19th national conference on Artificial intelligence* (AAAI'04), Anthony G. Cohn (Ed.). AAAI Press 755-760.

[3] Soo-Min Kim, Patrick Pantel, Tim Chklovski, and Marco Pennacchiotti. 2006. Automatically assessing review helpfulness. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing* (EMNLP '06). Association for Computational Linguistics, Stroudsburg, PA, USA, 423-430.

[4] Bing Liu Sentiment analysis and subjectivity In *Handbook of Natural Language Processing, Second Edition. Taylor and Francis Group, Boca, 2010*

[5] Jonathan Gemmell, Thomas Schimoler, Bamshad Mobasher, and Robin Burke. 2012. Resource recommendation in social annotation systems: A linear-weighted hybrid approach. *J. Comput. Syst. Sci.*78, 4 (July 2012), 1160-1174.