Predicting Usefulness of Yelp Reviews

Ben Isaacs, Xavier Mignot, Maxwell Siegelman

1. Introduction

The Yelp Dataset Challenge makes a huge set of user, business, and review data publicly available for machine learning projects. They wish to find interesting trends and patterns in all of the data they have accumulated. Our goal is to predict how useful a review will prove to be to users. We can use review upvotes as a metric. This could have immediate applications – many people rely on Yelp to make consumer choices, so predicting the most helpful reviews to display on a page before they have actually been rated would have a serious impact on user experience.

2. Data Set/Pre-processing

All of the data is made publicly available online[1]. All of the information is in JSON format, which made preprocessing relatively easy. Yelp provides a set of:

-42153 Businesses: These objects included information about the actual business being reviewed. Each contained features such as the average aggregate star rating, and the total review count. We also included a flag indicating whether or not the business was still operating.

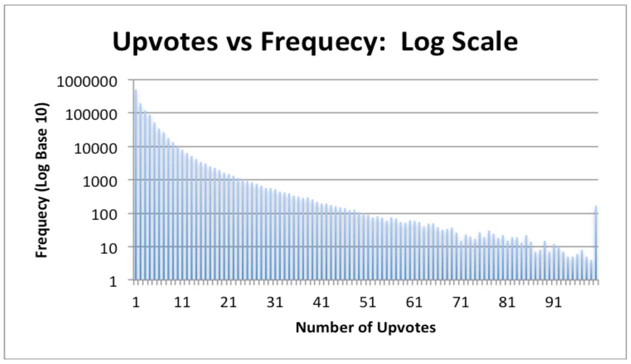
-252898 Users: Each user object provided data on the person posting the review, including a wide variety of information about the spread of other reviews that the poster had written previously. We included features such as average number stars, total number of previous reviews, time since joining yelp, and number of Yelp friends.

-1125458 Reviews: Each reviews is matched to a business and user. These became the actual training examples fed into each algorithm, after particular features were extracted and combined with the information about the user and business in question. Most importantly reviews were labeled with the total number of **useful, cool, and funny** votes – in aggregate, these served as the target variable.

Initially, we ran each algorithm with feature vectors containing only metadata. We also used text extraction on some of the reviews but the results did not seem promising enough to combine the two. So each training example consisted of:

{Review: Stars, Length of Text, Review Date}
{User: Review Count, Average Stars, Time Yelping, Number of Friends, Compliments(11 types)}
{Business: Average Stars, Review Count, Open/Not Open}

Text features were extracted using scikit learn's CountVectorizer on training example review texts. This approach extracts text features from a set of documents and returns token counts. We feed in all of the training example review texts (not the test data) and this makes up the vocabulary used. As noted above, text features proved pretty unhelpful so we decided not to append them to our final vectors.
Training examples were further normalized (zero mean, unit variance) using a built-in scikit learn functions. The same scaling values were used on the test data after training.
After preprocessing, one of the major challenges of working with this dataset was the spread of training examples. Many of the reviews (45.2%) had 0 upvotes, and the vast majority (98.6%) had less than 20. Even on a log scale, the distribution is heavily skewed:



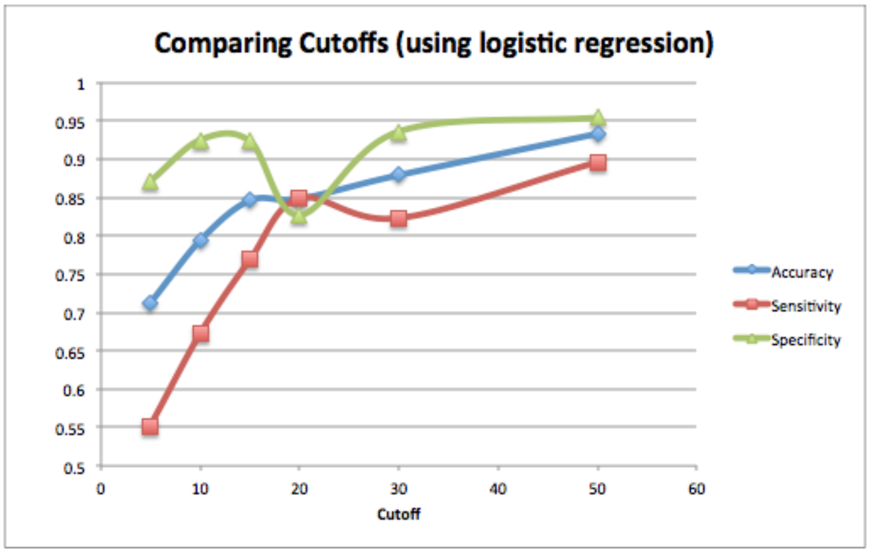
(Note the last bucket corresponds to 100+ upvotes)

3. Algorithms

We began by running linear regression to get a feel for the problem. However, it became apparent that due the distribution of the data predicting upvotes on a continuous scale through some sort of regression algorithm would be unlikely to produce valuable results. Linear Regression gave very little useful information, predicting 0 in most cases.

So we decided to reframe the problem in terms of a 0-1 classification, and then also as a several bucket prediction problem. We ran Gaussian Naïve Bayes, Logistic Regression, and SVM with a linear SVC kernel (all supported by scikit learn[2]).
SVM: The kernel function used was the inner product of the feature vectors:
$K(x^{(i)}, x^{(j)}) = < x^{(i)}, x^{(j)} > = x^{(i)}{}_1 x^{(j)}{}_1 + \ldots + x^{(i)}{}_n x^{(j)}{}_n$

To pick a good threshold count for number of upvotes we ran our set of algorithms with several different values (5, 10, 20, 50). Bellow is a plot of the testing accuracy at different thresholds:



With this in mind, we picked a threshold of 20 because we reasoned that a cutoff of 50 left us with too few positive training examples.

Additionally, we used bootstrapping to counteract the poor distribution of the data set. With bootstrap sampling we were able to get an idea of how effectively each of these algorithms could perform. We did this by creating 40 smaller datasets consisting of equal numbers of positive and negative training examples randomly pulled from the larger dataset. Random sampling was performed with replacement – i.e., the different datasets are not entirely independent. Then, each model was trained on all of the datasets, and the results were averaged across all 40 subsets. This allowed us to push each model to accurately predict positive examples rather than negative examples, which seems like a more interesting problem to solve.

4. Results – Important Features

The table bellow describes the results of each of these algorithms while considering a simple 0-1 classification. Score is a scikit learn function that represents accuracy for classification algorithms. For linear regression this was the R^2 coefficent of determination. In this case, bootstrapped means that the dataset used was a randomly selected subset of all the data containing an equal number of samples above the cutoff and below the cutoff. Note that negative values for the coefficient of determination are an artifact of the way it is calculated in the implementation we are using and can occur when a model is tested on data it was not trained on. Each of these was trained on 1000000 examples, and tested on 10000.

Accuracy results using metadata features	Training Score	Score on Random Test Sample	Score on Bootstrap Test Sample
Linear Regression	0.3661	0.3733	-0.032
Bootstrapped Linear Regression	0.3703	0.4163	-0.049
Gaussian Naive Bayes	0.9726	0.9782	0.7036
Logistic Regression	0.9879	0.9902	0.5501
SVM (SVC)	0.779	0.9058	0.5747
Bootstrapped SVM	0.9046	0.8909	0.83725
Bootstrapped Logistic Regression	0.8848	0.9496	0.848

Model Sensitivity and Specificity	True Positive Rate (sensitivity)	True Negative Rate (specificity)
Logistic Regression	.147	.99
Bootstrapped Logistic Regression	0.82	0.84
SVM	.11	.99
Bootstrapped SVM	0.84	0.83

For the review text vectors, we ran a Multinomial Bayes classifier again with a cutoff of 20 upvotes. Over 10 iterations, the average training and testing errors were:

NB average training score	83.99%

NB average test score	75.64%

Attempting to predict discrete bucket ranges was unsurprisingly less effective. We got the following results:

Recall for each Upvote Bucket	[0, 5)	[5, 10)	[10, 20)	[20, 50)	50+
Recall Rate	0.823	0.181	0.242	0.318	0.886

5. Discussion/Conclusions

We started by trying to run linear regression on the data because the number of ratings on a review is non-binary. However, we found that we got very poor results with that approach: our initial results yielded an R^2 score of 0.3661. We therefore decided discretize our output by using a single cutoff, where data above the cutoff is classified as positive and data below the cutoff is classified as negative. After testing around, we settled on a cutoff of 20 ratings for classification. With this cutoff, we ran Gaussian Naive Bayes, Logistic Regression and SVM algorithms on the data and achieved might higher success rates (full results shown above).

A big problem we had to deal with was data skew, as our logarithm histogram in our Data/Preprocessing section shows: most of the reviews have zero or close to zero ratings. As noted earlier we started training our models on even data sets (i.e., we took samples from the dataset such that each of our samples was half positively classified and half negatively classified). Bootstrapping, unsurprisingly, significantly improved the accuracy of our models, because the random samples we took were guaranteed to have an equal number of positive examples to train on. In particular, our true positive rates increased, which is a tradeoff that makes more sense in the context of this project since predicting 0 often garners much less useful information than correctly predicting which reviews will be popular. While models that were not trained on bootstrapped samples were superior at classifying random samples from the data, their low sensitivity caused them to perform very poorly when the test sets were constructed to contain an equal number of positive and negative examples.

In terms of our best classifiers, both logistic regression and SVM performed well. While our linear regression outperformed our baseline linear regression marginally, the negative score on the bootstrapped data indicates that it wasn't making any useful predictions (mostly negative predictions, as discussed above). However, our best rates significantly outperformed the baseline. Bootstrapped SVMs achieved a 95% success rate on random test data and an 85% success rate on bootstrapped data. By contrast, our baseline classifier (Naïve Bayes with a reduced feature set) scored 98% and 70% respectively. The massive increase in accuracy on the bootstrapped data represents a large increase in the usefulness of the model, since the ability to accurately predict positive examples is very valuable. A theoretical oracle that already had access to all the data would only outperform our bootstrapped SVM by 5% on the randomly sampled test set. However, 84% true positive and 83% true negative rates imply that there is still some room for improvement since an oracle with access to all the data would have perfect sensitivity and specificity.

The features we identified with the highest absolute weights were linked to how many of certain types of Yelp compliments the user had received. However, the feature that we identified as likely the most important was the number of Yelp friends the user had. Even though this feature often had a lower weight than some of the compliment features, the difference was fairly marginal (usually about .25). Further reinforcing the idea that the number of Yelp friends is the most important feature is that fact that most users do not have many compliments on their profiles. This means that in the majority of cases, the number of Yelp friends the user has ends up contributing the most to the decision the algorithm makes.

We were somewhat disappointed that the features with the highest weights were usually features related to user metadata, as discussed above. We had hoped to make predictions based primarily off the text in the review, but this approach was simply inferior to using the metadata as we can see from the relative accuracy rates above (75% versus 84%). This result has precedent. In our research for this project we came across a study of Twitter retweets, which also concluded that the most useful factor for determining retweets was the user who posted.

After reaching a degree of success classifying using our cutoff, we attempted to use the same learning algorithms (namely logistic regression) to classify the data into buckets. Rather than just predicting whether the review would have more or less than 20 upvotes, we attempted to predict what range the review would fall into. As expected, the accuracy of this method was significantly worse than what we attained with just the cutoff. However, the results are arguably more interesting since using buckets moves us closer to making actual predictions. We can see from the recall rates for each bucket that the algorithm was quite good at identifying examples that were on the high and low ends of the spectrum, which we hypothesize is because these reviews are more easily differentiable from the reviews in the middle of the spectrum. Some investigation indicated that this is probably mostly related to number of friends each Yelp user had, but this conclusion is complicated by the bucket sizes used. Further investigation could possibly have helped identify the exact point where the reviews become harder to classify and yielded insights into how these reviews are differentiated from the average review, but we had to leave this for future work.

6. Future

One obvious next step would be to implement this on new reviews as they come in (unlabeled) to provide a real testing set. It would also be interesting to see if more complex learning algorithms like neural networks could give better results then out of the box classifier implementations.

Additionally, more work could be done picking bucket ranges. These were arbitrarily chosen, and while we would always expect classification to outperform a bucket approach, work can be done to improve these predictions as well. We would also like to be able to do more investigation into how what makes certain review buckets easier or harder to predict, and identify cutoff points (in terms of number of upvotes) for where the reviews become hard or easy to predict.

7. References

1. "Yelp Dataset Challenge" *Dataset Challenge*. N.p., n.d. Web. 12 Dec. 2014. <http://www.yelp.com/dataset_challenge>.
2. Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011
3. Tauhid Zaman, Ralf Herbrich, Jurgen van Gael, David Stern. "Predicting Information Spreading in Twitter". December 12, 2014.   http://research-srv.microsoft.com/pubs/141866/NIPS10_Twitter_final.pdf
4. Jordan Segall, Alex Zamoshchin. "Predicting Reddit Post Popularity". <http://cs229.stanford.edu/proj2012/ZamoshchinSegall-PredictingRedditPostPopularity.pdf>