Stat 154 Final Project: Machine Learning Techniques to Predict Star Ratings of Yelp Restaurants

Bryan Alcorn	Paul Bramsen	Jong Ha Lee	Ying Luo	Vaibhav Ramamoorthy
SID 24393522	SID 24362392	SID 25344865	SID 24153624	SID 25539060
bbcorn	paulbramsen	leyldy	yingtluo	vaibhavram

May 5, 2017

Abstract

Yelp has anonymously open-sourced some of their data on restaurants and businesses. For our project, we sought to predict the star ratings of restaurants on Yelp using a wide range of features. These included business attributes (hours of business, etc.), check-in count, and, most importantly, a set of bag-of-words predictors that allow us to parse reviews for specific keywords. We prepared and joined the different datasets and then implemented an ensemble model that calculated a truncated mean of the predictions of 7 distinct models to result in our final predictions, which achieved a MSE of 0.32494.

1 Introduction

We wanted to apply the concepts that we have learned this semester in Professor El Karoui's Statistics 154 class in an interesting and valuable manner. For our final class project, we designed and created a model that allows Yelp to predict the star ratings of their listed restaurants, using a wide range of attributes on the restaurant, their users, and their reviews.

To Yelp, the ability to predict ratings for a restaurant given its attributes is invaluable. By gaining insight into which attributes (e.g. business attributes, user check-in trends, and review texts) influence positive and negative ratings, Yelp can, theoretically, provide businesses with suggestions to improve their Yelp ratings or filter restaurant search results for a particular user to boost specific restaurants that the user may enjoy more.

Overall, we were given data on 2,950 unique businesses, spread across 5 separate datasets.

(1) The **business** dataset consisted of 17 columns of data on 2,950 businesses, which are all Mexican restaurants, Chinese restaurants, pizza places, or coffee shops from one of 10 select cities. This dataset contained business attributes such as hours of operation, parking availability, credit card acceptance, etc. The businesses were split into 85% training set and 15% were in the test set. We were not given the star ratings for those in the test set.

(2) The **review** dataset contained all of the reviews written for the 2,950 businesses, which amounted to around 140,000—again split between train and test. It consisted of both review text as well as some ratings metrics (i.e. how many other users marked the review cool, funny, or useful).

(3) The **checkin** dataset contained checkin data for 2,877 of the 2,950 businesses, providing counts of hourly checkins for a monthly period.

(4) The **user** dataset consisted of user profiles of 88,914 Yelp users and included information on number of reviews written, compliments received across several categories, votes on reviews for several categories, and years in which the user was an elite Yelper.

(5) The **tip** dataset contained 27,301 tips written by users and included the number of likes given for each.

2 Initial Data Exploration and Preparation

2.1 Exploratory Data Analysis

We began by first conducting some exploratory analysis on the given datasets. From this, we found that the star ratings of restaurants in the training dataset of reviews roughly follows a normal distribution — that is, users tended to give restaurants 3 or 4 stars the most in their reviews. Due to this observation, we felt comfortable

designing our algorithms to predict star ratings at the review level and then averaging them to derive our finalized predictions for the star rating of the business itself.

2.2 Processing and Cleaning

Since several of the attributes were in array-string format, we implemented a text parser to convert these array-strings into distinct columns. This process was required for `business.hours`, `checkin.time`, etc. See Section 3.1 for more on this method.

After joining the data, we cleaned it by removing many of the predictors we thought would have no predictive value such as the `user_id`. We also tried keeping numeric predictors in place and getting rid of some of the categorical predictors that we didn't think would be as relevant to star rating of review. We hoped by doing this, our models would be easier to run, while being able to train all our models on data with meaningful and interpretable predictors.

We also had to convert some fields to numerics so that they could be included in models that could not handle factors.

2.2.1 Text Cleaning

As a first step in our natural language processing, we cleaned the review text before using it for feature engineering or predictive modeling through the following methods:

(1) Removed all punctuation marks from the reviews as well as stripping White Space from reviews, with the exception of apostrophes. This meant we may have ignored some important markers like "!" and ":)" but this was necessary in order to clean the text.

(2) Removed stop words that contained little sentiment, like "a" and "the."

(3) Removed numbers.

(4) Tokenized the words by lowercasing them.

(5) Lemmatized the words by grouping conjugations together. For example, "message" and "messaging" and "messages" are now all categorized under *messag*.

2.2.2 Data Joining

After all the data had been cleaned, we joined the data such that each row contains a unique review for a business. Our left join order was `((((business, review), checkin), user), tip)`. Most of the joins were on business_id or user_id, but tip was joined on `business_id`, `user_id`, and `date`. We chose to join on date as well, after observing that there were several edge cases where users would tip a particular business an exorbitant amount of times. This posed problems by making the join more difficult, so we chose to implement this simplifying assumption so that our data would not contain redundant rows.

2.3 Computing Resources

Due to the large amount of data we needed to incorporate into various models, we decided that we needed more compute power and memory than our personal computers offered. To this end, we rented a Google Cloud Platform Virtual Machine Instance. We ended up using about $60.00 worth of Google Cloud Platform credit (which was free thanks to Google's trial). The instance we chose ran Debian and had 4 dual core Intel Xeon CPUs @ 2.30GHz cores (so 8 cores in total) and 52 GB of memory. To make development easy, we installed R with RStudio server on our instance with an RStudio browser front-end. We created an account for each of your group members. This allowed anyone to easily log in and run large jobs on the server. To make data transfer easy, we set up a shared Dropbox directory on the VM and also wrote a cron job to automatically sync commits from our group's Git repo to the VM's local file system.

3 Feature Engineering Methods

3.1 Numerical Feature Engineering

Several of the predictors required numerical transformation in order to serve as features for our models. Some examples below:

(1) **business.hours:** We transformed the data on business operating hours into one-hot encoded features detailing which days the business was open and how many hours they were open each day.

(2) **business.attributes:** We constructed a list of all attributes present across all business in the dataset and then transformed the original field in to a one-hot coding for all attributes.

(3) **checkin.time:** We transformed the checkin time data into a series of fields that counted the number of checkins during the breakfast, lunch, dinner, and late hours, meant to paint a picture of the customer patterns of the restaurant.

(4) **user.compliments:** We aggregated the total number of compliments that the user received across all reviews.

3.2 Text Mining Techniques

3.2.1 Identifying Positive and Negative Words

Since text comprised a large portion in our models, we decided to mitigate the effects of "noise words", and amplify the effects of important words — that is, words strongly correlated with either positive or negative reviews. We decided to implement this with a Python script.

Our algorithm for doing so was as follows:

(1) For each word in each review, sum up the star rating for the review that that word is from.

(2) Divide by the number of times that the word appeared. This gave us a mapping from each word to the average star rating for each review the word appeared in, weighted by the number of times the word appeared in that review.

(3) Filter out words that appeared fewer than a cutoff number of times (to eliminate rare words that don't generalize—such as names—to essentially reducing overfitting), which we ended up choosing to be 25.

(4) Lastly, sort all words by weight.

This gave us a list of words that were most correlated with low star ratings at the bottom of the list and those that were most correlated with high star ratings at the top. We ended up deciding to incorporate the top and bottom 250 "most important" words into our sentiment analysis. As expected, words like *friendliest*, *mouthwatering*, and *brilliant* appeared in the list of positive words while words like *nastiest*, *incompetent*, and *vomiting* were among the most negative. Furthermore, common words that don't carry sentiment (e.g. pronouns, prepositions, etc.) like *the*, *and*, and *it*, were filtered out entirely as mentioned above. One interesting discovery we made was that reviews mentioning specific names (perhaps indicative of good customer service) tended to have more positive ratings.

3.2.2 Reviews Text Mining and Sentiment Analysis

From our tokenized bag-of-words matrix — with 116,474 reviews in the rows and 94,911 unique words within the reviews as the columns, and the elements were the frequency of the words in the given review — we conducted various text mining engineering techniques to create useful predictors.

In order to conduct sentiment analysis on the reviews, we first downloaded corpi of positive and negative words.[1] Then, we calculated positive and negative sentiment scores per review by calculating how many positive or negative terms — determined by the downloaded sentiment corpi — were in the review (also accounting for the frequency of these terms). Furthermore, we also gave double the weighting for words included in our own empirical sentiment corpi when calculating the sentiment scores to account for the specific data for this project.

In contrast to the bag-of-words model we initially utilized, we wanted to mitigate the effects of high dimensionality that would result if we included all the words as predictors. In doing so, we utilized a term frequency–inverse document frequency (`tf-idf`) weighting measure, which gives a numerical output to a word's "relative importance" to the review. After calculating the `tf-idf` weights for each word in each review, we only selected words (columns) which had a non-zero tf-idf weight in more than 2% of the reviews - meaning the word, in at least 2% of the reviews, had some sort of relative importance.

In conclusion, our text mining feature engineering technique on the reviews resulted in two main features: (1) Positive and Negative sentiment scores determined by word frequency, outsourced corpi, and empirical corpi, and (2) important words in reviews determined by `tf-idf`.

[1] https://www.cs.uic.edu/ liub/FBS/sentiment-analysis.html

4 Predictive Modeling and Ensembling

We trained 7 separate models on the joined version of the training set: bag-of-words, standard linear regression, random forest, bagging, boosting (XGBoost) using both classification and regression, and SVMs. If any singular prediction exceeded 5.0 or was below 1.0, we rounded down or up appropriately to ensure the prediction was within the prediction range. After running the models, we realized that bag-of-words and bagging were quite poor and so we chose not to include those in our ensemble. We averaged the predictions across the other 5 models, discarding the highest and lowest prediction to calculate a truncated mean, to arrive at the prediction for that specific review. Individual review predictions were then averaged across business_id to arrive at the predicted rating for the business.

4.1 Bag of Words

Initially we created a primitive predictive model solely using bag of words on all the review text in the training dataset. We then generated our first set of predictions by averaging the star ratings of all reviews for a given restaurant. As expected, this performed quite poorly relative to the other models. We did not include it in our ensemble.

4.2 Linear Regression

We performed an unregularized linear regression as our first model, including all predictors excluding `BusinessParking`, which turned out to be problematic since it contained a singular level throughout after populating NAs. Surprisingly, linear regression performed quite well in comparison to other models.

4.3 Random Forest

We implemented random forest classification by converting the star ratings to a factor. Our random forest model required additional data cleaning due to the lack of robustness of the `randomForest` package in R, including having to change some predictors such as n.activity to be numeric instead of a factor, and renaming variable names that started with numbers or contained reserved keywords like else and next. Random forest classification performed well, with one of the lowest test set MSEs out of all of the models.

We were unable to perform random forest regression due to lack of computational power (machine kept crashing). See Section 6.1 for greater explanation.

4.4 Bagging

We hoped that bagging would help us produce a more stable predictor from the data. After testing Random Forest, we got an MSE reading of .19, but when we submitted to Kaggle, it was above .35. This instability made us consider using bagging to help improve the stability of our predictor so that when we submit, we would have more confidence in what our MSE results would be. Unfortunately the MSE is quite high for bagging and after some trial and error, we decided not to include it in our ensembled model.

4.5 Boosting

Since we had 537 predictors, we figured that instead of choosing the few most important predictors, another approach would be to build many trees/classifiers with different predictors, and aggregate them in an effective way to create a good predictive model. Thus, we incorporated a boosting model via eXtreme Gradient Boosting in R, in both classification and regression settings - both performed quite well.

4.6 Support Vector Machine

Lastly, we implemented a Support Vector Machine model, since we had learned in class that is one of the more powerful models, especially when kernelized. We chose to implement one of the most common kernels, the radial basis kernel, since it is very widely used. The SVM output turned out to be relatively worse than some of our other aforementioned models, but we nonetheless chose to include it in our additive ensemble because it was still one of our better models.

5 Results

To estimate the training MSE of our results, we divided our joined training dataset into 80% training and 20% test sets and applied our trained model on the 20% test set, calculating the MSE for each. These MSEs were

calculated on the review-level rather than the business-level, so it does not make sense to compare it to Kaggle scores, but rather should be used as relative measures to compare individual models. Listed below are the results for each of our individual models:

Method	Training Mean Squared Error
Bag of Words	1.2281
Linear Regression	0.7196
Random Forest (Classification)	0.1970
Bagging	0.9462
Boosting (Regression)	0.6010
Boosting (Classification)	0.6980
SVM	0.7599

We then used each trained model to predict the review star ratings of the test set and averaged the predictions of these individuals models in our ensemble. The resulting MSE (on 50% of the testing set on Kaggle) was 0.32494.

6 Conclusion

For our project on Yelp's dataset, we were able to build a well performing model based on reviews and business attributes. We learned how to apply basic natural language processing techniques to find and utilize patterns among good and bad reviews, as well as how ensembling multiple models can be effective for making more accurate predictions. We can envision Yelp using similar machine learning techniques to extract specific insights about where businesses can make improvements. And as mentioned before, Yelp can benefit from these findings by being able to offer more services and suggestions to these businesses.

6.1 Limitations

Though we did rent a powerful Google Cloud Platform server to enable us to process larger quantities of data and complex models, there were still certain models that demanded more memory and compute than we had. For example, after multiple crashes and days of waiting we decided to give up on random forest regression model, despite trying nearly every available R random forest package. If we had had more time, it would have been nice to prune the dataset and try alternate libraries to see if we could have succeeded with these models.

We were also unsuccessful with latent variable models. Initially, we thought they might be a nice way to get initial star rating predictions; however we soon realized that this model was best for guessing what a user would rate a restaurant when you already have ratings form some other users for said restaurant. This data was not available for our use. In practice, this is a very powerful model since it allows Yelp (or companies with a similar rating system) to predict what users will like and use this information to generate recommendations.

Another limitation was our use of time and technical experience. We could have applied more advanced natural language processing or other models like neural nets, but given our time constraints, we were not able to add these models. As a future step, we would want to explore creating better features through a more thorough review of our predictors and extracting better insights from the information given.

6.2 Future Analyses

6.2.1 More Natural Language Processing Techniques

Ultimately, the only natural language processing model we had incorporated in our prediction models was bag of words. This meant we only examined how word frequency played into star ratings, and thus there were many sentiments that we had missed out on in review texts. For example, if a review contained the text "wasn't terrific," our bag-of-words model would have captured "terrific" — our model would be more biased toward a higher rating, when in fact this phrase lends itself to a more negative sentiment. For this reason, we would recommend that future studies look into the n-gram model because this would deeply enrich our understanding of the review sentiments.

6.2.2 Incorporation of Other Predictors

In addition to using more advanced techniques, we would also recommend looking into incorporating some of the other available Yelp data. Specifically, we had not used the short tips that reviewers may leave for other people about particular restaurants in our predictive models. We decided to leave these out because our models were already taking so much computational power as is, and we had felt that tips were not as informative in comparison to reviews. However, it is possible that, with text mining and natural language processing techniques, analyzing tips could have led to a significant improvement in our star predictions.