University of California, Los Angeles

Yelp Review Prediction Model
Final Report

Olivia Apuzzio (605201571), Kellen Whetstone (105408164), and Bruno Romani (705382868)

STATS 101C

Professor Shirong Xu

4 Dec. 2022

Section 1

1.1 Introduction

We use the data set provided by Yelp as part of their Data Challenge 2014 for training and testing the prediction model. Concretely, the dataset consists of three files, one for each object type: review, user, and business. Each file consists of one json-object-per-line. Thus, a review is represented in the "review.json" file as a json object which specifies the user ID, business ID, stars (integer values between 1-5), useful (users who see this review can pin on this review to see if it is useful), cool (same description as useful, but to pin if a review is cool), funny (same description as useful, but to pin if a review is funny), and review (actual review from user with text). The user's review count is represented in the "user.json" file as a json object which specifies the user ID, user review count, user useful count, user funny count, user cool count, elite (year since user became an elite member), user fans, and user average star rating. The business average star rating is represented in the "business.json" file as a json object which specifies the business ID, state, city, and business average star rating. The necessary data for our project is contained in the review.json, user.json, and business.json files.

In this paper, we treat the Review Rating Prediction problem as a binary classification problem in Machine Learning, where the labels are either "good" or "bad" reviews. Good reviews are composed of reviews with 4 or 5 stars, and bad reviews are composed of reviews that are composed of a rating of 1-3 stars. When we initially looked at the data, we found that the distribution of the reviews is skewed negatively, meaning there were considerably more high ratings. Because of this, we decided to classify three star ratings as "bad" reviews. Since there was still a skewed distribution, we fixed this issue by oversampling the dataset to make it balanced. More details on this in the implementation section.

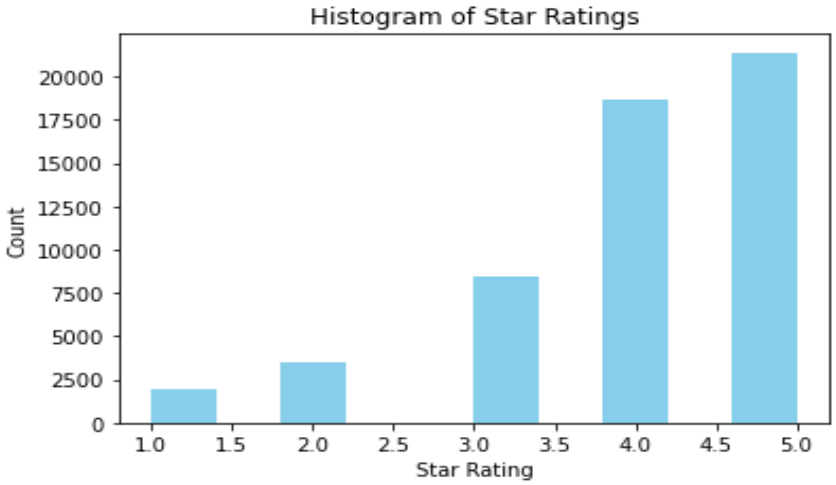


Figure 1, Histogram of Star Ratings in Dataset

We use two supervised learning algorithms - random forest and logistic regression. We build two prediction models in total. We train and evaluate the performance of each of these models on the data set provided by Yelp. The rest of the paper is organized as follows. Section 2 provides the details of preprocessing and feature extraction on the dataset. Section 3 presents the supervised learning algorithms we used for our model. Section 4 provides detailed results and analysis of our models. To conclude, section 5 provides concluding remarks and future work.

Section 2

2.1 Preprocessing

For the final dataset provided, all three json files were attached together. However, since the data set was excessively big, we decided to add some parameters that can help us create not only a smaller dataset but also one that makes sense to apply a model to.

To begin with, we only use data from the state of California. This removes data points from other states or cities included in the raw data set that are not part of California. As for user specific, we only utilized data points of users who have reviewed more than 150 businesses. We ended up with a final dataset of 53845 rows (reviews) and 18 variables. To put the data into perspective, this data set contains the reviews of 11204 different users, 3530 different businesses in 25 different cities, and 53845 reviews.

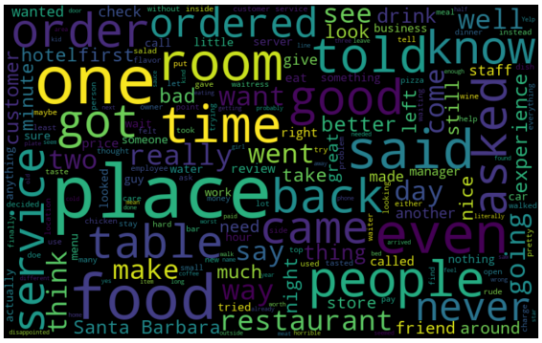
With the final dataset, we continue to clean and preprocess the data, specifically the reviews. To accomplish this, we first created a new column as a duplicate of the reviews, which we used to clean and preprocess the data. Because Yelp reviews may contain special characters, stop words like “it”, “the”, “is”, etc, along with other syntax that is generally useless in characterizing rating, it is necessary to remove this jargon so we can extract the meaningful content. We utilized standard Python libraries to remove stop words, along with single characters and special characters like parentheses. Additionally, we substituted multiple spaces with a single space. After cleaning the data, we now were able to move on to extraction.

2.2 Feature Extraction

In order to pick out which words were most common for good and bad ratings, we first checked for the ten most frequent, and ten most infrequent words across the entire dataset. Next, we normalized the words by making each character lowercase, which we were able to do after first tokenizing the words. To separate the good and bad reviews, we created a “label” column, by assigning a one to four or five star reviews, and a zero to anything else. We made a conscious choice to include three star reviews as bad, because the data was skewed fairly positively, which left us with considerably less low reviews.

We also created visualizations in the form of word clouds, in order to get a better representation of the words associated with certain ratings. A word cloud is a visualization method wherein the most frequent words appear in large size, and the less frequent words appear in smaller size. Pictured below are the word clouds for the 4-5 star reviews (left), and 1 star reviews (right).

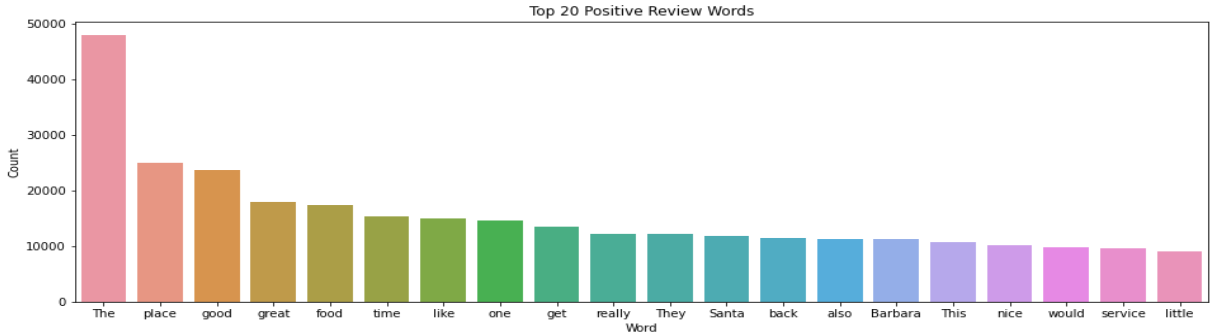
- **Figure 2,** Word Cloud 4-5 Star Reviews



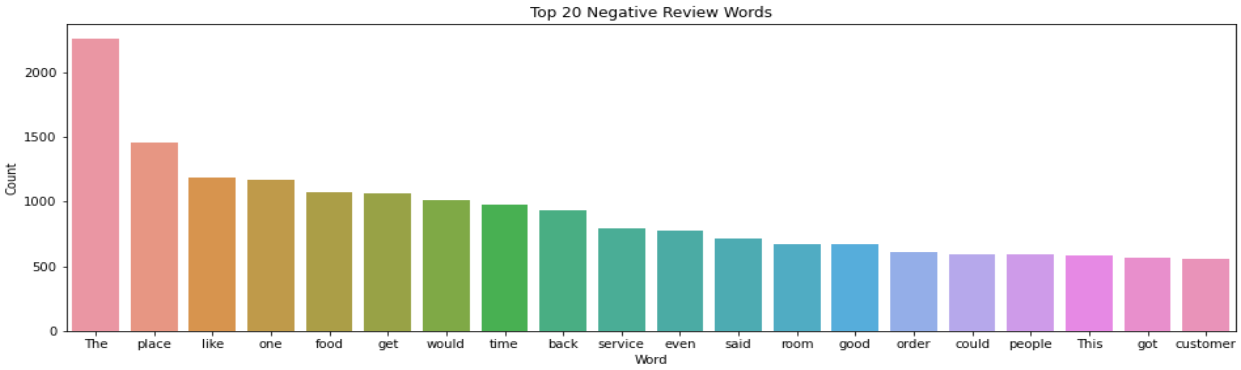
- **Figure 3,** Word Cloud 1-3 Star Reviews

Furthermore, we obtained the average sentiment score using sentiment analysis from TextBlob. The sentiment property of TextBlob returns a namedtuple of the form polarity and subjectivity. The polarity score is a float within the range (-1.0, 1.0), where -1 defines a negative sentiment and 1 defines a positive sentiment. The subjectivity is a float within the range (0.0, 1.0) where 0.0 is very objective and 1.0 is very subjective. We found the sentiment score for good reviews to be 0.3052, while the sentiment score for bad reviews was -0.0045, which indicates the good reviews had a much more positive sentiment. Interestingly, the "bad" reviews were just slightly below 0, which was a bit unexpected given a negative sentiment score is closer to -1.

Additionally, we plotted high frequency words for both good and bad reviews, with the results pictured below.



- **Figure 4**, Top 20 Positive Review Words Barchart



- **Figure 5**, Top 20 Negative Review Words Barchart

As you can see in the plots, the top 20 positive review words included many with a generally positive connotation, such as “good”, “great”, and “nice”. On the other hand, the top 20 negative review words only contained one word with a positive connotation, that being “good”. This gives us a general idea of the words we should be extracting for our algorithms.

We decided to use the “bag of words” algorithm for our feature extraction. Bag of words is an algorithm within the Sklearn library that transforms text into fixed-length vectors, allowing us to evaluate similarities between reviews. We use the function “CountVectorizer” to grab the unique words in the reviews, and then transform our result into a matrix with review number as the rows, and words as the column. We used the command stop words, to eliminate the meaningless words in the reviews. The matrix is stored in the “bow” variable, and we can use this matrix to fit our models.

Section 3

3.1 Supervised learning

To train our prediction models, we use two supervised learning algorithms - random forest and logistic regression.

3.1.1 Random Forest

We chose random forest because its efficiency is notable in large datasets and this one is a large dataset. We preferred to utilize this over other algorithms. Unlike other models, it doesn't tend to overfit with a lot of features, which again we have a large amount of features here. For classification tasks, the output of the random forest is the class selected by most trees, which would be {1 : “Good”, 2 : “Bad”}. For regression tasks, the mean or average prediction of the individual trees is returned.

3.1.2 Logistic Regression

We chose logistic regression because of its ability to predict a binary outcome based on its independent variables. For our purposes, it will be predicting the binary outcome {1: “Good”, 2:“Bad”}- given its independent variables applied in the model. Given a new feature vector for a new review, we compute our model with the given new vector to get the corresponding binary outcome value.

3.2 Performance Metrics and Implementation Details

First, we apply an oversampling method in order to have a balanced dataset since it is currently skewed with more “good” reviews. Since we have increased the number of bad reviews to an equal amount of good reviews, the dataset is balanced and we can proceed with training and testing. We use 70% of the dataset for training, and 30% for testing.

Then, we apply a randomized search to choose optimal hyperparameters for our model. For the random forest model, we determined the best hyperparameters to be the number of estimators is 800, which is the number of trees used in the random forest. The minimum sample split is 10, which is the minimum number of samples required to split a node. The minimum sample leaf is 1, which is the minimum number of samples at the leaf node. Last hyperparameter is max depth at 25, which is the depth of the tree.

For the logistic regression model, we determined the best hyperparameters to be solver set as liblinear, which is the algorithm used in the optimization problem. The class weight set is none, since our data set is balanced. The inverse regularization is C = 0.3002, with the default value being 1. The smaller the inverse regularization value is, the stronger the regularization is.

For each of both of the prediction algorithms, we perform 3-fold cross validation on the training set and compute two metrics, precision and accuracy.

All of the implementations are done on a Macbook Pro M1 CPU with 4 cores (1.6 Ghz each), 8 GB RAM and 64bit MAC operating system. The programming language used is Python, and extensive use is made of its libraries such as numpy and scikit-learn.

Section 4

4.1 Results and Analysis

4.1.1 Random Forest Model Performance

To begin with, when we run the final random forest model with the best estimator, we evaluate the fit of the model by looking at the test score, train score, accuracy, precision, and recall score. As for our train score, we attained 97.07% of correctness, and as for our test score,

we attained 86.24% of correctness. Using a confusion matrix, we were able to calculate the accuracy of our model. We obtained an accuracy score of 83.37%, which shows the model performs reasonably well given that the data was balanced with an oversampling method. The precision came out to be 89.30%. The recall score of our model was 88.12%, which is a good score to show that the model is able to classify positive results very effectively.

4.1.2 Logistic Regression Performance

For our logistic regression model, when we run the logistic model with the best estimator, we evaluate the fit of the model by looking at the test score, train score, accuracy, precision, and recall score. As for our train score, we attained 99.36% of correctness, and as for our test score, we attained 87.18% of correctness. Using a confusion matrix, we were able to calculate the accuracy of our model. We obtained an accuracy score of 82.94%, which shows the model to have a slightly worse accuracy than the model we implemented earlier with random forest. The precision was 90.10%. The recall score was 86.47% for the logistic regression model, which was slightly worse than the 88.12% recall score for the random forest model. Additionally, given the slightly higher accuracy score of the random forest model, the random forest model is likely the better choice.

	Predicted	
Actual	Good	Bad
Good	2043	884
Bad	996	7385

- **Figure 6**, Logistic Regression Confusion Matrix

	Predicted	
Actual	Good	Bad
Good	2132	795
Bad	1134	7247

- **Figure 7**, Random Forest Confusion Matrix

Models	Predicted Test Score	Train Score	Accuracy	Precision	Recall Score
Random Forest	0.862	0.971	0.834	0.893	0.881
Logistic Regression	0.872	0.994	0.829	0.901	0.865

- **Figure 8**, scores of both models. The first row is random forest, and the second row is logistic regression.

Section 5

5.1 Conclusion

After being presented with this dataset, our goal was to determine the similarities in words across Yelp reviews, in order to build a model that could accurately predict whether a review was negative or positive. We developed a procedure to accomplish this goal, which included preliminary analysis and preprocessing of the data, feature extraction, training of our prediction models using random forest trees and logistic regression, and finally implementation of our models and analysis. We evaluated the fit of our models with test score, train score, and accuracy. In the end, we believe the better model was the random forest model, given its slightly better accuracy over the logistic regression model. However, the logistic regression model is also an appropriate model for the prediction of a “good” or “bad” review.

5.2 Future Remarks

While we were able to fit an adequately performing model, it is important to analyze our procedure for future discovery. To this point, it may be interesting to perform future analysis by recharacterizing three star reviews as a “mediocre” review, treating our review rating predictions as a multi-class classification problem. In our prior analysis, we treated rating predictions as a binary classification problem, which meant three star reviews were included in “bad” ratings with one and two stars. Creating another classification may help narrow our conclusions and allow us greater flexibility in prediction.

In addition to performing future analysis, it may be interesting to apply both, oversampling and undersampling, methods to the data set to have a more balanced data set with less bias in creating new bad reviews with only using an oversampling method on our dataset. Applying only an oversampling method leads to possible cause of overfit on the minority class. Therefore, an application of both methods may be interesting to see how much more accurate the models can be.

References

Gupta, A. (2022, August 23). *Python: Linear regression using sklearn*. GeeksforGeeks. Retrieved December 8, 2022, from https://www.geeksforgeeks.org/python-linear-regression-using-sklearn/

Qiao, F. (2019, January 8). *Logistic Regression Model Tuning* . Towards Data Science. Retrieved December 8, 2022, from https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5

Singh, H. (2021, April 6). *Random Forest Algorithm: Random forest hyper-parameters*. Analytics Vidhya. Retrieved December 8, 2022, from https://www.analyticsvidhya.com/blog/2021/03/introduction-to-random-forest-and-its-hyper-parameters/

Xu, S. (2022, November 10). *Lecture 8 Random Forest*. Random Forest Application. Retrieved 2022, from https://drive.google.com/file/d/1E3WQCNtDl90WkBtcdy_7uJyJRRMvWEPu/view?usp=sharing

Xu, S. (2022, November 3). *Lecture 7 Text Mining*. Applications of Text Mining. Retrieved 2022, from https://drive.google.com/file/d/1kBP6fmi38cG3fAbRBuz1sNU-KzL0z9zJ/view?usp=sharing

Xu, S. (2022, October 6). *Lecture 3 Logistic Regression*. Application of Logistic Regression. Retrieved 2022, from https://drive.google.com/file/d/11um0dommb7O_0wft9JDjcNL_FVsl9lOi/view?usp=sharing