Springboard Data Science Career Track

Capstone Project 1 Final Report

A Sentiment Analysis of Yelp Reviews

Walker Page

2020

1. Introduction

Yelp is an online platform that, among other things, allows customers to post reviews and ratings of businesses to describe their experience with those businesses. This platform is a valuable tool for customers because it enables them to make more informed decisions about which businesses to use and which to avoid. Yelp has the potential to be invaluable for business owners as well, and this project makes progress towards developing one way that Yelp could be improved as a resource for business owners. The project focuses specifically on restaurants.

For restaurants to thrive as businesses it is important for them to know whether their customers have a positive or negative experience. Restaurant owners might acquire this knowledge in a number of ways, but one method is to read Yelp reviews that customers write about their experiences. A problem with this method, however, is that it involves conflicting incentives for restaurant owners. On the one hand, reviews are a valuable source of information about customer experiences, so restaurant owners should want more customers to write more reviews. On the other hand, reading and interpreting reviews can be time-consuming, and the more reviews there are, the more time it takes to gain value from them. Furthermore, it can be difficult to gain an overall sense of customer experiences simply by reading a set of reviews. For example, the combination of reading seven shorter positive reviews and a single extremely long and negative review might skew the impression someone gains of overall customer experience.

The goal of this project is to provide a more efficient means of discriminating between positive and negative reviews. In particular, the goal has been to construct models that can reliably classify reviews as positive or negative based on the content of the reviews. Having models that can do this not only removes the problem of needing to read through each review just to get some sense of customer experiences, but it also provides a way to get a clearer view of overall customer experience. For example, after classifying a set of one-hundred reviews, even before reading any of the reviews, a restaurant owner could easily figure out whether a majority of the reviews are positive or negative, and if so, how much of a majority. If Yelp added a feature that enabled restaurant owners (and eventually any business owner) to easily assess the

sentiment of individual reviews as well as overall customer experience, the platform would be a much more useful resource for owners.

For this project, I have construed the business problem as a supervised classification problem, and I have developed two models to classify reviews, both of which perform well.[1] In what follows, I will describe my approach, summarize my findings, draw conclusions from those findings, and make recommendations for Yelp moving forward.[2]

2. Approach

2.1 Data Acquisition and Wrangling

The data I used for the project includes approximately 6.5 million reviews of a variety of different businesses.[3] I acquired the data directly from Yelp, but it needed to be manipulated in several ways to be useful for the present project.[4] First, since the dataset contained reviews for many different kinds of businesses, and the present project is concerned only with restaurants, I removed reviews for businesses that were not restaurants.

Second, I narrowed the size of the dataset by removing the columns that were unimportant for this project. Although the removed columns represent variables that might be useful in a larger context, since the present project is focused on the polarity of the reviews, I decided to remove these columns to simplify the analysis. For example, I removed columns of data that included items like the name and address of the business. This step left three columns of

[1] The two models can be viewed here: <https://github.com/walkerpage/springboard/blob/master/Capstone_Projects/Capstone_Project_1/Code/Capstone_Project_1_Baseline_Modeling.ipynb> and here: <https://github.com/walkerpage/springboard/blob/master/Capstone_Projects/Capstone_Project_1/Code/Capstone_Project_1_Extended_Modeling.ipynb>

[2] All code for the project can be found in the following repository: <https://github.com/walkerpage/springboard/tree/master/Capstone_Projects/Capstone_Project_1/Code>

[3] For the documentation for the dataset, see here: https://www.yelp.com/dataset/documentation/main.

[4] The dataset was acquired here: https://www.yelp.com/dataset.
The notebook in which I clean the data can be found here: https://github.com/walkerpage/springboard/blob/master/Capstone_Projects/Capstone_Project_1/Code/Capstone_Project_1_Data_Wrangling.ipynb

data: ‘business_id’, which is a unique identifier for each restaurant, ‘stars’, which includes the star ratings associated with the reviews, and ‘text’, which includes the text of the reviews.

Next, I inspected the dataset to see how many rows contained either duplicate or missing values. There were no missing values in the dataset, and I eliminated the duplicates.

Finally, after closer inspection, I discovered that some of the reviews were not written in English. Since analyzing non-English reviews was beyond the scope of the project, and the number of non-English reviews was negligible relative to the size of the dataset (only around 24,000 out of over 4 million total reviews), I removed them from the dataset as well.

2.2 Storytelling and Inferential Statistics

After acquiring and cleaning the data, I explored a number of interesting features of the data.[5] In particular, I analyzed (1) properties of the star rating data, (2) the review data, and then (3) the interactions between those two sets of data. Regarding the star rating data, I investigated the total number of each star rating (see Figure 1), the proportion of each star rating out of all ratings (see Figure 2), the average star rating for each restaurant (see Figure 3), and the distribution of ratings for some specific restaurants that have multiple reviews.

[5] The notebooks in which I explore an analyze the data can be found at these two links: https://github.com/walkerpage/springboard/blob/master/Capstone_Projects/Capstone_Project_1/Code/Capstone_Project_1_Data_Storytelling.ipynb, and https://github.com/walkerpage/springboard/blob/master/Capstone_Projects/Capstone_Project_1/Code/Capstone_Project_1_Statistical_Data_Analysis.ipynb

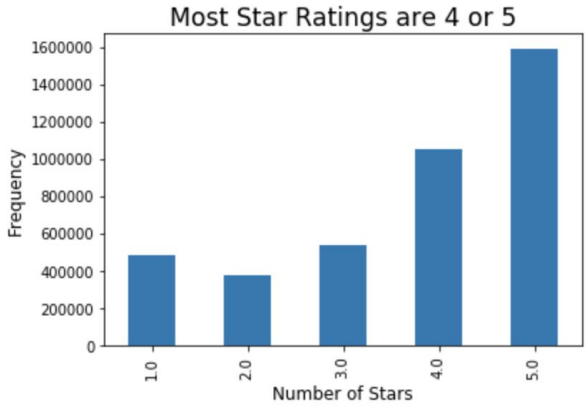

Figure 1

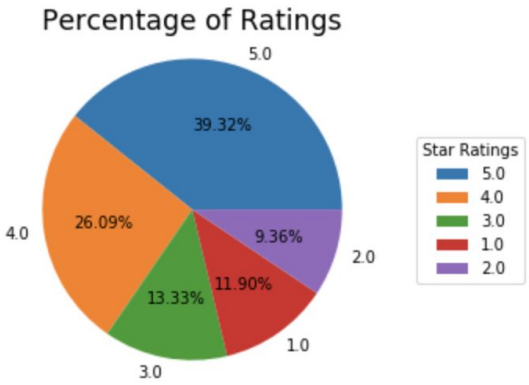

Figure 2

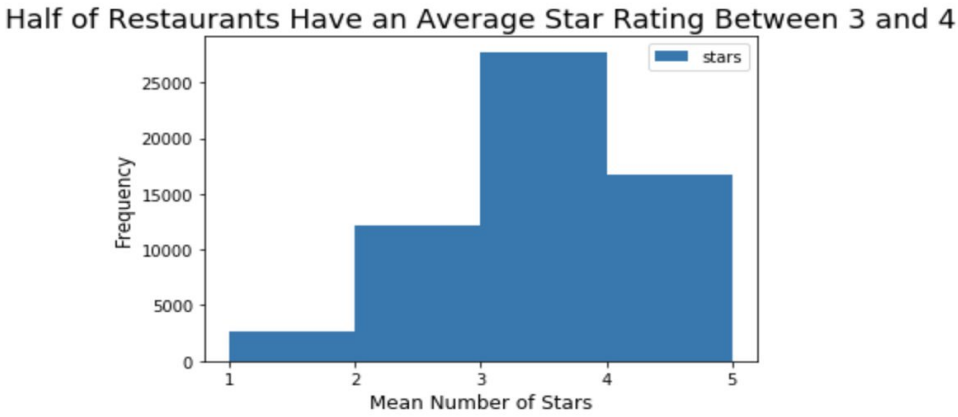

Figure 3

For the review data, I explored the number of reviews for each restaurant in the dataset and the distribution of lengths for the reviews (see Figure 4).

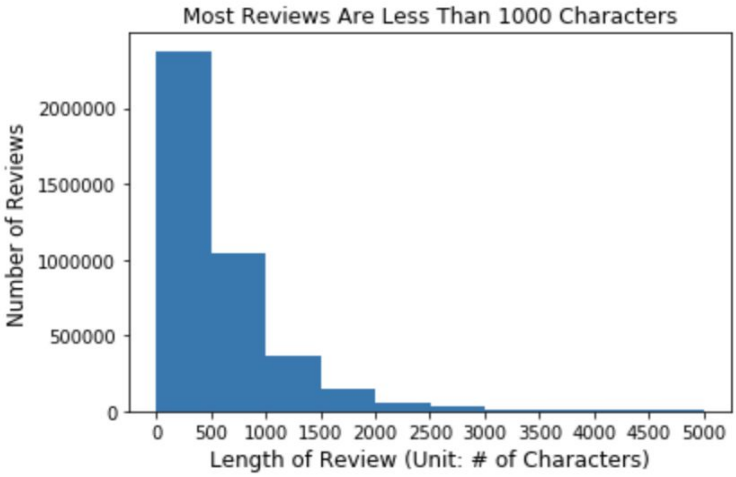

Figure 4

Finally, in order to explore how the star rating data interacts with the review data, I identified the relationships between the different star ratings and lengths of reviews. In particular, I explored the mean and median length of reviews for each star rating (see Table 1, Figure 5, and Figure 6). I also created word clouds to depict words that occur frequently in the reviews at each star rating level, but I have not included these word clouds in the report because they were not particularly informative. Many of the most frequent words appeared the same in the word clouds across different star ratings.

Star Rating	Mean Length
1.0	669
2.0	734
3.0	702
4.0	616
5.0	469

Table 1

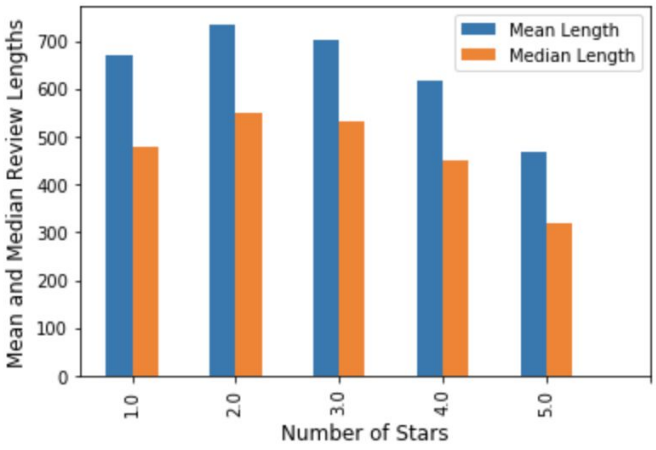

Figure 5

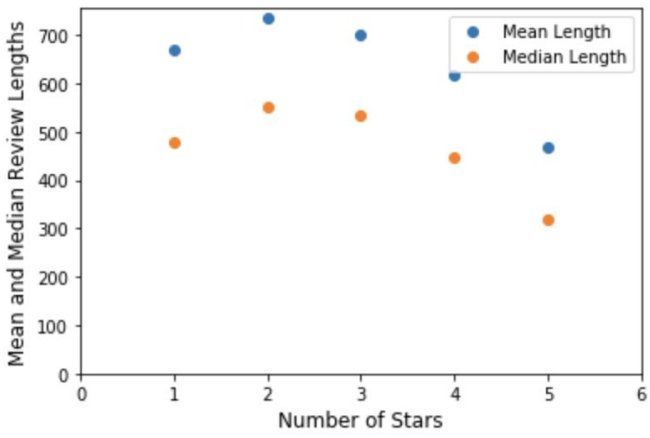

Figure 6

One of the more interesting findings from this exploration was that the average review length (measured in terms of mean character count) differed across each of the star ratings. Since this feature could potentially be used in the classification models, I decided to use hypothesis testing to determine whether these differences in average length were statistically significant. First, I performed a one-way analysis of variance (ANOVA) to test whether there were any statistically significant differences between the mean lengths of reviews for the different star ratings. The null hypothesis was that the mean review lengths are the same for each star rating, and the alternative hypothesis was that at least one of the mean review lengths differs from another. The test produced an F-statistic of 36.22 with a p-value approximating 0, which is

considerably smaller than the threshold I used of alpha = 0.05. The test thus provided significant reason to reject the null hypothesis and conclude that at least one of the mean review lengths differs from another in the population.

Since the ANOVA test can only tell *that* one of the mean lengths is significantly different from another mean length, and not *which one* is significantly different, I subsequently conducted several t-tests (using the Bonferroni correction to control for familywise error) between pairs of star ratings to determine which differences in mean length were statistically significant. In particular, because when I trained the models, I associated star ratings of 1, 2, or 3 with negative sentiment, and star ratings of 4 or 5 with positive sentiment, the most important comparisons for my purposes were between these two groups of ratings. Accordingly, I performed t-tests of the difference in mean review length between the following six pairs of star ratings: 1 & 4, 1 & 5, 2 & 4, 2 & 5, 3 & 4, 3 & 5. The t-tests revealed that the differences between nearly all of these pairs were statistically significant. The only difference that was not statistically significant was between the 1 and 4 star reviews. The analysis suggested, therefore, that the length of a review could potentially help to classify it as positive or negative. I did not ultimately use this feature in my models because one goal of the project was to improve model performance as much as possible while also keeping the models as simple as possible, but (as I suggest below) it could be useful to include this feature to improve model performance even more in future developments.

2.3 Baseline Modeling

After completing the above exploration and analysis, I constructed two different classification models.[6] As a baseline, I developed a Logistic Regression model. Then to extend the modeling I developed a Random Forest model. Before constructing the two models, however, I prepared the review text by making all words lowercase, removing punctuation, and removing a list of stop words. These are standard text pre-processing techniques used to improve the quality of the models. Making all words lowercase ensures, for example, that 'great' and 'Great' will not

[6] The two notebooks in which I develop the models can be found at these two links: https://github.com/walkerpage/springboard/blob/master/Capstone_Projects/Capstone_Project_1/Code/Capstone_Project_1_Baseline_Modeling.ipynb, and https://github.com/walkerpage/springboard/blob/master/Capstone_Projects/Capstone_Project_1/Code/Capstone_Project_1_Extended_Modeling.ipynb

be treated as distinct words, and removing punctuation has the same effect for pairs of text like 'great...' and 'great'. Stop words are words that occur frequently in written English and consequently have proven to be generally unhelpful for discriminating the sentiment of a body of text. Stop words, therefore, can tend to create noise in the model, and performance is typically enhanced by removing them.

After processing the review text in these ways, I constructed a vector representation of the text, using a bag-of-words approach as it is implemented in the Scikit-Learn method CountVectorizer().[7] On this representation, reviews are modeled as a collection of so-called documents, where each review constitutes a document. The unique words extracted from the documents compose a vocabulary, and this vocabulary is represented as a vector of words. The vocabulary thus constitutes a vector of features extracted from the documents, and these features can be used to build a matrix, X, that has as many columns as the length of the vocabulary, and as many rows as there are documents. Moreover, each column in the matrix is an individual feature (i.e. word) from the vocabulary, and each row is a vector representation of an individual document (i.e. review). The distinguishing characteristic of the CountVectorizer() method is that the values of the matrix in a row, *i*, and column, *j*, correspond to the frequency (i.e. 'count') of the *j*-th feature (i.e. word) in the *i*-th document (i.e. review). Additionally, associated with matrix X there is a column vector y with the same number of rows as X, and is such that the value of y at row *i* is 1 if the review represented at row *i* in X is 'positive', and 0 otherwise.[8]

With this (X, y) representation in place, I began to develop the baseline Logistic Regression model. I tested four different versions of the model with each varying along two dimensions. First, the versions differed with respect to whether the division between training and test data was stratified. When the division is stratified, the proportion of the two classes (positive and negative reviews) within the test data is made to be approximately the same as the proportion

[7] Documentation for CountVectorizer():
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
On the notion of a bag-of-words approach, see here:
https://machinelearningmastery.com/gentle-introduction-bag-words-model/.

[8] Note, strictly speaking CountVectorizer() produces a sparse matrix representation, which does not store zeros explicitly.

of the two classes within the training data. When the division is not stratified, this proportion is not necessarily preserved.

The second dimension along which versions of the model differed was with respect to whether they used L1 or L2 regularization for the 'penalty' parameter of the model. These two forms of regularization shrink the coefficient estimates of the underlying linear model, thereby reducing the model's variance.[9]

The four different versions of the model that I tested are summarized Table 2.

Version 1 ● No Stratification in train-test-split ● L2 Regularization for fitting the model	Version 2 ● No Stratification in train-test-split ● L1 Regularization for fitting the model
Version 3 ● Stratification in train-test-split ● L2 Regularization for fitting the model	Version 4 ● Stratification in train-test-split ● L1 Regularization for fitting the model

Table 2

After tuning the model's hyperparameters in each version, it turned out that Version 4 performed the best. Some specific performance metrics of Version 4 of the model are discussed in section 3 below.

2.4 Extended Modeling

In the extended modeling stage of the project, I explored a number of variations on the Random Forest model. I also investigated how two different ways of vectorizing the review text influenced the model's performance. In the ScikitLearn implementation, these two methods for vectorizing text are called CountVectorizer() and TfidfVectorizer().[10] As I explain above, CountVectorizer() produces a matrix where the columns are all of the individual terms in the

[9] For a helpful overview of L1 and L2 regularization and how they differ from ordinary least squares, see section 6.2 in James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An introduction to statistical learning: with applications in R* (7th ed.). New York: Springer.

[10] See the above footnote for a link to the documentation for CountVectorizer(). Here is a link to the documentation for TfidfVectorizer():
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

vocabulary, the rows are individual documents, and the matrix values are the count or frequency with which the term in the column occurs in the document at that row. TfidfVectorizer() produces a similar matrix but term frequencies within individual documents are weighted by the frequency with which they occur across all documents. Words with a smaller frequency across documents are more discriminative and thus are given greater weight. In the final analysis, using CountVectorizer() turned out to have a slight performance advantage.

In addition, I explored all combinations of three other parameters of the Random Forest model. First, I tried using values ranging from 50 to 150 for the number of decision trees (a.k.a. The number of estimators) in the model. Although increasing the number of trees does improve performance, I found that increasing the number of estimators beyond 100 did not produce a significant performance advantage.

Second, I explored three different specifications for the number of features to consider when looking for the best split as the model was fit: the total number of features in the data, the logarithm of the total number of features to the base 2, and the square root of the total number of features. I found that using the square root as the maximum number of features for determining the best split generally performed at least as well, if not better than using the total number of features. And both of those options performed better than using the logarithm.

Figure 7 and Figure 8 represent the differences in the OOB error rate of the model when modulating the kind of vectorization, the number of estimators/trees, and the number of features.

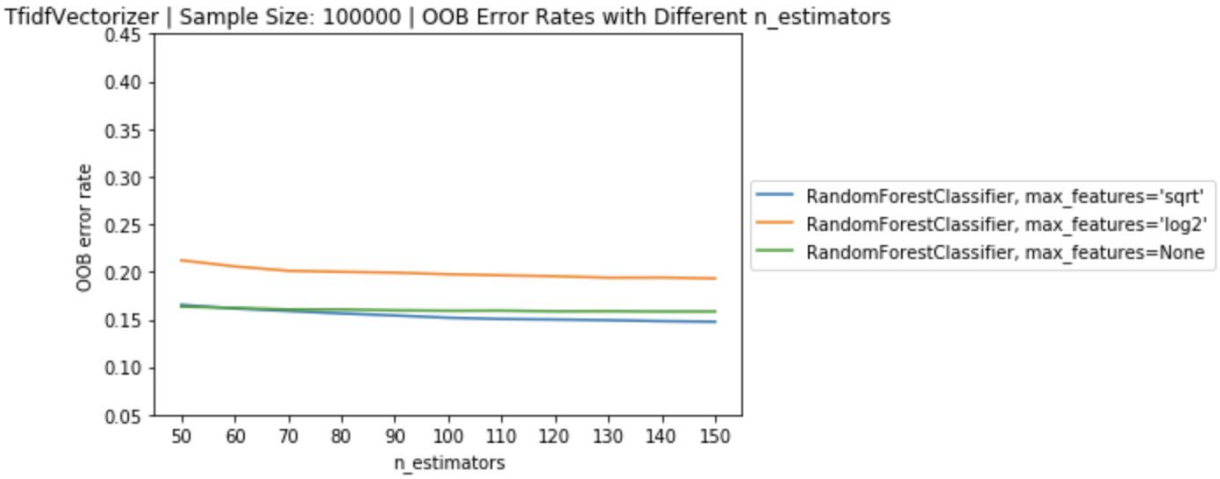

Figure 7

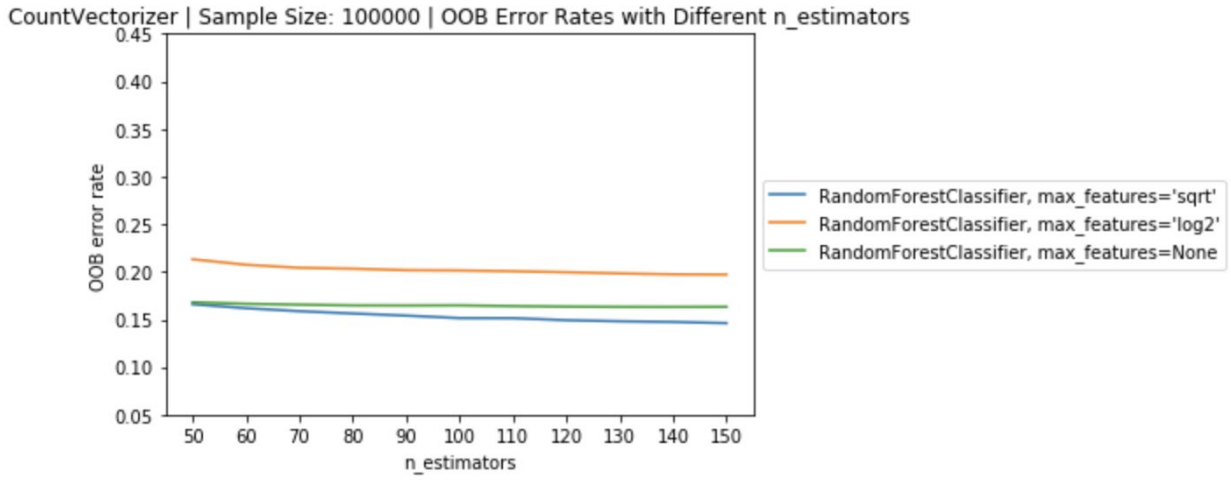

Figure 8

Third, and finally, I explored two different measures of node purity–typically referred to as 'gini' and 'entropy'–which are used to evaluate the quality of splits as the decision trees in the forest develop.[11] It was discovered that using the gini metric generally improved performance over the entropy metric.

To summarize the results of my analysis, I found that the best performing variation on the Random Forest model had the following features:

- Relied on a vector representation of the text constructed using CountVectorizer()
- Used 100 estimators/decision trees
- Used the square root of the number of features as the maximum number of features when fitting the model
- Used the gini index as the measure of node purity

In the next section, I will present some of the specific performance results of the model with these properties.

[11] For a helpful overview of how these measures differ, see Ch. 8 in James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An introduction to statistical learning: with applications in R* (7th ed.). New York: Springer.

3. Summary of Findings

Overall, both models (Logistic Regression and Random Forest) performed very well. Table 3 summarizes key performance metrics for each model.

Performance Metrics:	Logistic Regression	Random Forest
Accuracy	0.90	0.87
Precision	Positive Class: 0.90 Negative Class: 0.89	Positive Class: 0.85 Negative Class: 0.90
Recall	Positive Class: 0.95 Negative Class: 0.81	Positive Class: 0.96 Negative Class: 0.68
F1-Score	Positive Class: 0.92 Negative Class: 0.84	Positive Class: 0.90 Negative Class: 0.78

Table 3

The same test data was used when testing each model. The data included 300,000 observations, and it was divided between 103,656 observations in the negative class and 196,344 observations in the positive class. Accuracy is the percentage of correct predictions out of all predictions made by the model. Precision, Recall, and F1-Score are all relative to the two classes. Precision is the percentage of reviews correctly assigned to a class out of all the reviews assigned to that class, so Precision tells us how precisely the class labels were assigned to the reviews by the model. Recall is the percentage of all reviews belonging to a class that were correctly assigned to that class, so Recall tells us how many instances of a class the model correctly recalls. The F1-Score is the harmonic mean of precision and recall. As is clear in the table, the Logistic Regression model out performed the Random Forest model with respect to almost every performance metric, which suggests that Logistic Regression is the best algorithm for Yelp to use.

4. Conclusions and Future Work

Overall, this has been a successful project, but there are a number of directions that could be worth pursuing in future work. The following suggestions are worth exploring:

- Using stemming or lemmatization in the text preprocessing.[12] Using these techniques would change the number of features in the data, and hence could affect model performance.
- Using n-grams (for different values of n) as the features in the vector representation of the text rather than simply individual words. As above, this technique would change the number of features in the data, and hence could affect model performance. It could improve the model's ability to evaluate difficult reviews where, for example, a positive term is paired with a negation (e.g. 'the food was not great').
- Adding additional features, such as the length of the review,[13] location of the business, or demographics of the reviewer (which might be obtained via the US Census Bureau), to help with class prediction. Again, adding features can affect model performance.
- Using different values for the minimum or maximum document frequency for a word to be included as a feature. For example, if a word only occurs in a single document or if it occurs in every document, then it may not be very helpful for discriminating between positive and negative reviews. Hence, specifying a minimum document frequency above 1 and a maximum document frequency below the number of documents could improve model performance.
- Trying to split the data into training and test sets *before* creating the vector representation of the text.
- Using other algorithms (e.g. Naive Bayes, Support Vector Machines, Neural Nets, etc.) to develop alternative models.

---

[12] See here for a helpful explanation of stemming and lemmatization: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[13] It was concluded in my statistical analysis that the differences in review lengths between a number of different star ratings are statistically significant. My statistical analysis can be viewed in the notebook here: https://github.com/walkerpage/springboard/blob/master/Capstone_Projects/Capstone_Project_1/Code/Capstone_Project_1_Statistical_Data_Analysis.ipynb.

- Develop an ensemble of different models that have been built. Ensemble approaches often outperform isolated models.

Despite these unexplored directions, it is encouraging that the models I constructed perform so well.

5. Recommendations for the Client

Yelp is a useful resource for customers, but it has a lot of undeveloped potential for business owners. This project has shown just one feature that Yelp could add to make its platform more useful for business owners, and I have made significant headway towards developing this feature. My central recommendations for Yelp moving forward are to:

- Continue refining the Logistic Regression model I have developed
- Explore some of the directions mentioned in the last section

With the resources available to Yelp, and its access to large amounts of data, the model's performance can undoubtedly be improved beyond its current status, which would make the suggested feature even more useful for business owners.