Sentiment Analysis of Yelp Reviews: An Accuracy Comparison of Naive Bayes and Logistic Regression

Bryan Ton
Group 8
Santa Clara University
bton@scu.edu

Derek Hu
Group 8
Santa Clara University
dhu1@scu.edu

Maile Naito
Group 8
Santa Clara University
mnaito@scu.edu

ABSTRACT

In the current digital age, online reviews have become a substantially important aspect for businesses and manufacturers. Not only can online crowd-sourced feedback impact sales of a certain product, it can also affect the success of an entire business. For example, if a restaurant has low ratings on a social media feedback platform, such as Yelp, it is likely that these ratings will influence a future potential customer's opinion. Yet, due to this large amount of unstructured data, each review cannot be analyzed manually and must use some variation of automatic analysis. In this paper, we focus on the sentiment analysis of Yelp reviews by using classifiers based on the Naive Bayes and logistic regression algorithms. These classifiers are used to determine whether a review is positive or negative. Through experimental evaluations, we concluded that both classifiers had a similar maximal accuracy by using various techniques such as removing stop words and weighting terms.

1 INTRODUCTION

With the growth of the internet, the use of online feedback systems has become an important part of the success of a business. Ultimately, businesses care about a customer's opinion. With these feedback systems, other customers are also able to view the opinions of prior customers and base their initial judgement on a specific item or service. This is important because now a customer no longer has to purchase a product in order to make a decision of whether they will make another purchase or even recommend it to their friends. With an online system, businesses and customers alike have access to helpful feedback. For customers, the feedback serves to answer the primary question: is it worth it to buy this product? If a certain product has good reviews, a customer is more willing to purchase the product; if a product has negative reviews, a customer will most likely purchase a different product with higher reviews. For businesses, the use of this feedback varies. Positive online reviews left by customers may have a significant impact on the growth and success of a business by building a good reputation, while negative online reviews may serve as a source of criticism to be used for the improvement of the business.

These online feedback systems generate large amounts of data that make it unfeasible to perform manual analysis of each individual review. Along with a text review, most feedback systems use a numbered score (such as "five-star rating" system), which makes it easy to produce an average score as a basis of judging. But for companies that are looking to improve their businesses based on the feedback from these sites, these numerical ratings do not provide any meaningful information: a business cannot specify what

contributes to low ratings or what part of their business customers enjoy the most. Thus, an approach known as sentiment analysis is used in the industry to perform an automated process of classifying these text reviews based on the sentiment, attitudes, and emotions of the text. This approach allows for an analysis of the correlation between certain words and a specific rating.

In this paper, we will focus on the reviews of restaurants on the online social platform, Yelp. The problem that we focus on in this paper is what algorithms and techniques are the most effective and accurate in classifying a review as having either positive or negative sentiment. In this project, we focused specifically on comparing the Naive Bayes and logistic regression algorithms and analyzing their respective results when used to classify the dataset. The objective of this project was to observe the outcomes of classifiers based on each algorithm and determining which preprocessing techniques were the most successful. Naive Bayes served as our main algorithm, while logistic regression was used as a baseline comparison algorithm. We discuss the related work that we reviewed before implementing this project, then describe in detail how each classifier works. We then present our procedure of implementing the classifiers, as well as the preprocessing techniques used. In our procedure and evaluation, we provide illustrative examples and visualizations to provide a means of better understanding of the dataset and classification results. We conclude with reflections on the project, ways to improve our work, and where the future of sentiment analysis is headed.

2 RELATED WORK

In 2002, a similar comparison study was conducted by Andrew Y. Ng and Michael I. Jordan from University of California, Berkeley. The study compared logistic regression and Naive Bayes as a means of comparing discriminative and generative classifiers, respectively. Their paper discusses another study that concluded logistic regression and Normal Discriminant Analysis (another form of generative learning) was only marginally more statistically efficient. Their own study showed that depending on the size of the dataset, logistic regression is able to show a much more noticable difference in performance. On larger datasets, logistic regression was able to outperform Naive Bayes, while for smaller datasets, both algorithms performed similarly.

3 METHODOLOGY

For our project, we focused on the Naive Bayes and logistic regression algorithms. Naive Bayes served as our primary algorithm of focus, and logistic regression served as a baseline comparison algorithm. The two algorithms both share an independence of features assumption, which makes it easier to compare the results between the two classifiers. Each classifier uses a different approach to obtain its probabilities: Naive Bayes uses generative learning, while logistic regression uses discriminative learning.

3.1 Naive Bayes

The Naive Bayes classifier (Choudhari and Dhari, 2017) is a simple probabilistic classifier based on Bayes theorem. The classifier uses the joint probabilities of words and class labels from the training set to find the probability of a class label. The algorithm is naive because of its strong independence assumption between each feature. In text

analysis, it assumes that each word's impact on a class label is independent of all other words in that instance (which in this case, is the review sentence).

The Bayes theorem for finding probability is defined as:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Where:

- B is given feature (i.e. word)
- A is a class (i.e. A = positive or negative)
- P(A | B) is Posterior Probability
- P(B|A) is Maximum Likelihood/ Conditional Probability
- P(A) is Class Prior Probability
- P(B) is Predictor Prior Probability

Posterior Probability is the probability of class label A occurring given feature B in the test set. Conditional Probability is the probability of feature B belonging to the class label A in the training set. Class Prior Probability is the probability of class label A occurring in the training set. Predictor Prior Probability is the probability of feature B occurring in all the class labels in the training set.

We decided to implement Naive Bayes because of its multiple advantages. It is a simple algorithm that does not require a large amount of data. It trains and classifies data much quicker than other text classification algorithms. Even though it is based on independent feature vectors and their probabilities, the classifier tends to perform fairly accurately.

Some disadvantages of Naive Bayes to be aware of are the Zero Observation problem and its independence feature assumption. The Zero Observation problem is when a word occurs in the test set that is not present in the training set, then the classifier automatically assigns that word's probability as 0. The independence feature assumption assumes that each word does not correlate with the occurrence of other words, which, in some cases, is very impactful on the success rate of a classifier.

3.2 Logistic Regression

The Logistic Regression classifier (Bishop, 2006) is a linear model used to classify instances, usually for binary dependent values. The model uses a logistic function to calculate the probabilities of the possible outcomes. In our implementation of logistic regression, the following cost function was used (along with its minimization):

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Where:

- X is a vector of inputs
- w is a vector of coefficients
- c is the lower bound for the cost function

The cost function can be modified/ varied depending on the dataset and its features. The cost function allows for the data similarity to be analyzed as an optimization problem.

We decided to implement logistic regression because it is considered a generative classifier and produces its probabilities differently from that of Naive Bayes. The logistic regression classifier shares similar advantages to that of Naive Bayes, such as it is easy to implement and is efficient to train. An additional advantage of logistic regression is that it is robust to noise, which greatly reduces the likelihood of overfitting.

Some disadvantages of the algorithm are the independence feature assumption, sensitive feature selection, data load limit, and reliance on transformations. Similar to the Naive Bayes algorithm, logistic regression assumes independence among all its features. For logistic regression, feature selection is

very important because choosing features that are not as meaningful can impact the classifier's accuracy. The classifier also does not handle a large amount of features very well, and relies on transformations for nonlinear features.

4 EXPERIMENT

4.1 The Dataset and Preprocessing

Our Yelp Review dataset came from UC Irvine's Machine Learning Repository. It comes in the form of a simple text file with 1000 lines, where each line is it's own review that ends in either a 0 or a 1, signifying that the review was positive or negative. In order to maximize our ability to correctly classify these reviews, we must first understand the dataset that we are working with. By manually inspecting the dataset, we see that each review is about a sentence long (ranging from 5-15 words) and labelled with either a zero or one, representing a negative or positive sentiment, respectively. Some of the reviews contained misspelled words, as well as words with special characters, such as é used in the words Café, fiancé, and puréed. Since we were able to observe these characteristics of our dataset, we were able to already have an idea of what techniques could possibly be used to reduce the number of incorrect classifications.

To avoid the negative impact of misspelled words, we decided to use stemming (for example, in the case of where the last letter of the word was repeated, although this is a specific case of a misspelling). Stemming also served as an effective approach since each review was short and if we were to consider every word as a singular token, then it would be more difficult to determine the most meaningful features (in this case, which words are the strongest predictors of a positive or negative sentiment). On the flip side, we removed stop words that seemed to appear often in both positive and negative reviews.

For example, the word food had a large effect on both the positive and negative word cloud, so we turned it into a stop word. We also removed words that seemed to have nothing to do with the quality of the restaurants being reviewed. In this case, we removed the word Vegas, or the stem vega, which appeared fairly often and yet only described where the reviews were, not necessarily the quality of the restaurant. Once our dataset was properly stemmed and stopwords were removed, we began feeding them into our two classification algorithms.

Figure 1: Positive Sentiment Word Cloud

Figure 2: Negative Sentiment Word Cloud

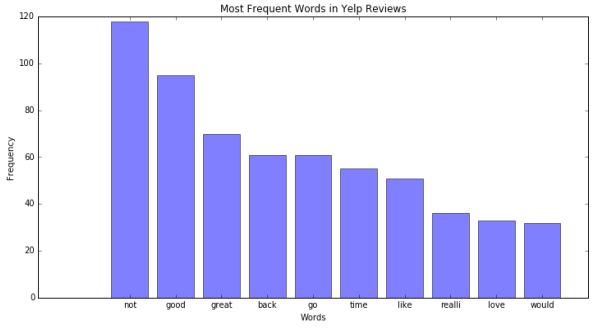

Figure 3: Most Frequent Words in Dataset

4.2 Naive Bayes

To implement our Naive Bayes classifier we used the NLTK and TextBlob libraries for Python. In both of these libraries the Naive Bayes classifier was already written into functions that we could use. Our task was to put our dataset into a workable format for the function to use.

Using the Open function in Python we were able to read in the text file and form our dataset into a list of tuples, each entry in the list contained a review and its associated rating. With our data in this format we could begin to use the tools in NLTK to begin cleaning out text. First we used NLTK's word tokenizer to separate each word in every review. With all the words, we could compare them against our list of stopwords and remove any matches. The initial list of stopwords we use came from NLTK, but we then modified it to remove any punctuation and various words such as "vegas" and "food" which we deemed to be irrelevant to the sentiment analysis. We also included a regular expression to filter out any words with punctuation, namely various contractions. Any words that were not stopwords were then stemmed using the Porter stemmer stem function. Stemming was used to combine all the forms of our words together and condense their counts.

With our dataset of 1000 reviews we found that a 300-700 test-training data split yielded the best accuracy. Initially we were using a 50-50 split, but we later reasoned that having a higher quantity of training data would give the classifier more to work with and yield better accuracies. We then tested this by running each split from 100 to 500 test data in increments of 50 to find that 300-700 was the optimal split with our implementation.

To train our classifier we simply had to run our training data through the NaiveBayesClassifier function from TextBlob. We could then take the classifier and run data through it using the classify function to obtain the classifier's classification of any kind of review. To obtain the accuracy of our classifier, we ran our test data with the accuracy function which would take all the classifications and compare them to the actual ratings to obtain a percentage of how many it got correct. Through our implementation of Naive Bayes, we were able to obtain an 82% accurate classification of the test data.

After we obtained this result, we then put our findings into various plots to help visualize how well our classifier did. We could also use some of these visualizations to compare the classifier to Naive Bayes to determine which may be optimal for sentiment analysis.

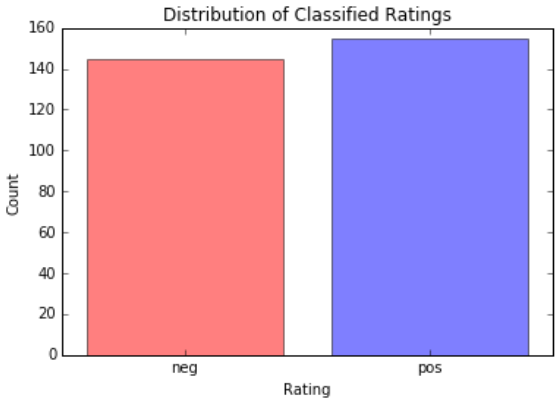

Figure 4: Classified Sentiment Distribution, Naive Bayes

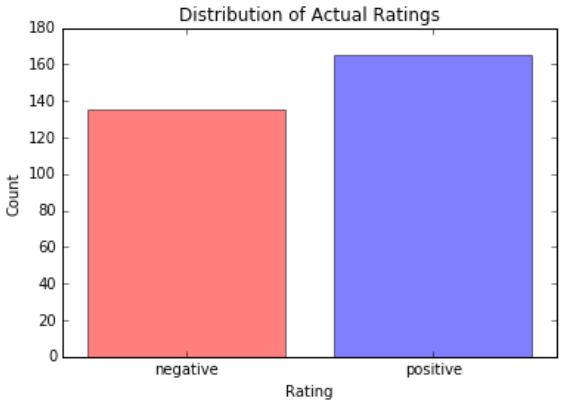

Figure 5: Actual Sentiment Distribution, Naive Bayes
Note: Figures 4 and 5 are distributions of the test set that returned the best accuracy

4.3 Logistic Regression

To implement logistic regression, we utilized Pandas as well as the Sci-Kit Learn Machine Learning tool set for Python. Sci-Kit Learn provides its own logistic regression function, which we used to create our logistic regression model. In order to do that however, we had to first transform the reviews into a form that the function could read it.

Using Pandas, we first split up the reviews into two categories, reviews, which held the actual reviews, and ratings, which held the actual ratings, either positive or negative. Since Sci-Kit Learn also provides it's own CountVectorizer within its feature extraction class, we used that to vectorize our data into a feature vector, which we can then transform and feed into the logistic regression model. We decided that the best way to split up our data was to split it up evenly in half. But rather than just top bottom, we did every other as training data and every other as testing data. In this way, we can split the data more evenly while also making sure that there are no parts or patterns in the review are missing in the testing/training data.

We used a Term Frequency - Inverse Document Frequency transformation for Logistic Regression. Taking advantage of the Sci-Kit Learn tf-idf transformer also in its feature extraction class, we can easily transform both our training and test data to be fed into the predict function given to us by the Logistic Regression model. In doing so, we achieved an accuracy of about 80%.

In an attempt to improve our accuracy, we decided to apply k-fold cross validation to our logistic regression model since it already ran quickly. In order to do this, we used the Sci-Kit Learn Pipeline tool, which streamlined the process of tokenizing and transforming our reviews into a simple line or two of code. Afterwards, we split our data into k folds and then apply K-Fold Cross Validation. After some testing, we decided that 50 folds returned the best accuracy of about 83%.

After getting our results, we visualized them as a probability distribution of reviews as well as a distribution of classified and actual ratings. We used these visualizations alongside our results as a way to compare our success to the Naive Bayes algorithm.

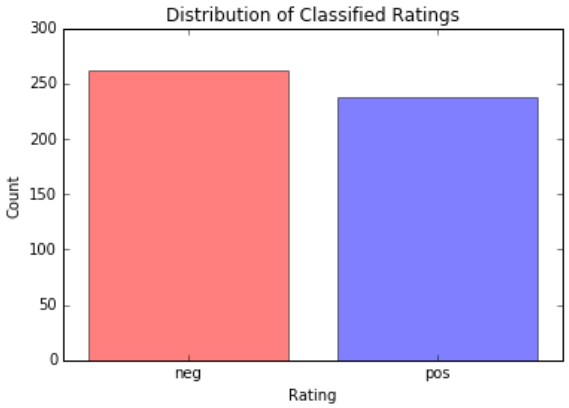

Figure 6: Classified Sentiment Distribution, Logistic Regression

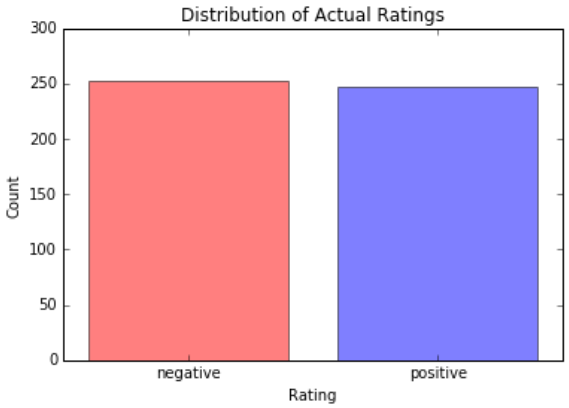

Figure 7: Actual Sentiment Distribution, Logistic Regression

Note: Figures 6 and 7 are distributions of the test set that returned the best accuracy

5 EVALUATION

Our method led to a good accuracy score for both the Naive Bayes and Logistic Regression algorithms. However, the results from 'From Group to Individual Labels using Deep

Features' returned even better results. So in comparison, ours did not perform as well, scoring about 5% -10% below their accuracy scores.

The original study was performed on 3 different datasets, Amazon, IMDb, and Yelp, which differs from ours which only focused on the Yelp reviews. As such, we ended up being a little bit more limited in the data that we could test our implementations on. However, they applied multiple different levels of Logistic Regression but as a way to compare to their own algorithm, Group-Instance Cost Function. As it turns out, their Bag of Words model on Documents with Logistic regression ended up giving the best results for the Yelp Reviews, which scored almost 10% higher than ours. This is most likely due to their tokenization and stemming process, which was a lot more detailed and careful than ours. While we used a general vectorization, stemming, and stopword removal, they applied weights to their words and used word embeddings.

Therefore, there is much more that can be done to improve our experiment to be both more accurate and more detailed. Some things that we could do is apply multiple different variations of our algorithms in order to test not only against other algorithms, but to test against how it performs under different data processing methods. For example, the original study did Bag of Words on both documents and sentences, as well as embeddings on documents and sentences.

Our only measure of success was also only our accuracy score. We compared the accuracies of the two algorithms by using normalized confusion matrices to visualize the accuracies of the correct classifications. While accuracy is one of the most important indicators of success rate for these algorithms, the study also uses Area Under the Curve as another indicator of success for their algorithm. Implementations of measuring Area under the curve and possibly false positive/negative measure could lead us to a better idea of what our algorithms are successful in.

One of the major flaws of our original execution was that we did not tokenize our data in the same way for both algorithms. This leads to results for two different sets of data, which is useless for comparison since the parameters and details of the review data are fed in differently. Since we ended up with accuracy scores that are not comparable, our results end up leading to much less useful conclusions than if we were to feed the same tokenizations into our algorithms.

	Accuracy			AUC		
	Amazon	IMDb	Yelp	Amazon	IMDb	Yelp
Logistic w/ BOW on Documents	85.8%	86.20%	**91.25%**	88.08%	88.32	**94.41**
Logistic w/ BOW on Sentences	88.3%	81.81%	78.16%	87.19%	82.67	67.87
Logistic w/ Embeddings on Documents	67.82%	58.23%	81.00%	61.24%	60.77	82.59
GICF w/ Embeddings on Sentences	**92.8%**	**88.56%**	88.73 %	**91.73%**	**88.36%**	92.36%

Figure 8: Accuracy and Area Under the Curve Scores for 'From Group to Individual Labels using Deep Features'

```
Wow... Loved this place.
        Positive: 96.74%; Negative: 3.26%

Crust is not good.
        Positive: 18.07%; Negative: 81.93%

Not tasty and the texture was just nasty.
        Positive: 8.45%; Negative: 91.55%

Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.
        Positive: 97.28%; Negative: 2.72%

The selection on the menu was great and so were the prices.
        Positive: 99.95%; Negative: 0.05%
```

Figure 9: Probability Distributions Naive Bayes

```
Crust is not good.
        Positive: 15.19%; Negative: 84.81%

Stopped by during the late May bank holiday off Rick Steve recommendation and l
oved it.
        Positive: 86.75%; Negative: 13.25%

Now I am getting angry and I want my damn pho.
        Positive: 14.84%; Negative: 85.16%

The potatoes were like rubber and you could tell they had been made up ahead of
time being kept under a warmer.
        Positive: 32.23%; Negative: 67.77%

A great touch.
        Positive: 98.27%; Negative: 1.73%
```

Figure 10: Probability Distributions Logistic Regression

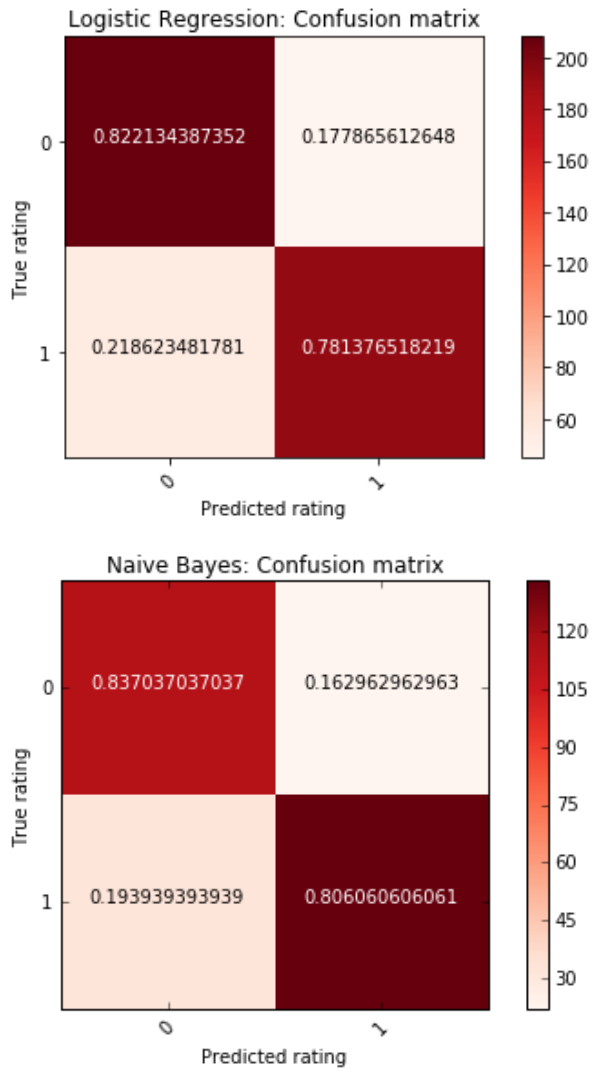

Figure 11: Normalized Confusion Matrices of Naive Bayes and Logistic Regression's Classified Reviews

6 CONCLUSION

Our work has shown us that Naive Bayes and Logistic Regression, in our current implementation, perform fairly similarly in terms of accuracy. The direction of this work now is to look for ways to improve our implementations of these algorithms. The overall goal of testing these algorithms, is to maximize the ability to correctly classify reviews in order to process large quantities for surveying. The drawback though, is that there end up being false positives and negatives, which we want to limit. Some of the ways to fix them are to implement stop word removal on words that have significance in both positive and negative reviews. The removal of typos and correct interpretation of homonyms could also lead to a higher success rate.

Some steps for possible improvement in our work itself could be to also apply more Machine Learning Algorithms such as SVM as another algorithm to compare our results to. This could give us a larger baseline and better understanding of what leads to success in machine learning algorithms.

Once we reach a point where we can say that our algorithms are successful, the next step would be to find a way to measure what details lead to a more positive or negative review on yelp, pinpointing the qualities of a successful and unsuccessful business.

REFERENCES

Bishop, C. M. (2006). Pattern recognition. Machine Learning, 128, 1-58.

Choudhari, P., & Dhari, S. V. (2017). Sentiment Analysis and Machine Learning Based Sentiment Classification: A Review. International Journal Of Advanced Research In Computer Science, 8(3), 1051-1056.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011, June). Learning word vectors for sentiment analysis. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1 (pp. 142-150). Association for Computational Linguistics.

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. Advances in neural information processing systems, 2, 841-848.

Kotzias, Denil, De Freitas, Smyth, (2015) From Group to Individual Labels using Deep Features

10 APPENDIX

Appendix A
Task Distribution

To divide up the project, we decided to split up the coding portion. Derek was in charge of implementing the Naive Bayes classifier and Bryan and Maile would implement the logistic regression classifier. After the two were created, we would get together and compare the results of the two and record our findings. We decided it would be best to get together and collaborate on the paper to combine all that we learned from our project.

Appendix B
System Requirements and Libraries

The code and algorithms used in this project were implemented on Unix and Windows systems. While it was not tested on other systems, ideally it should be able to run on them given Python and the necessary libraries are installed. The required libraries are NLTK, Numpy, TextBlob, word_cloud, and scikit-learn. To install these, open terminal and enter the following line by line:

```
sudo pip install -U nltk
sudo pip install -U numpy
pip install -U scikit-learn
pip install -U textblob
pip install wordcloud
Python -m
textblob.download_corpora
sudo python
import nltk
nltk.download()
```

A window should appear to download nltk packages. Choose "`/usr/share/nltk_data`" for the download directory, go to the "All Packages" tab and install "stopwords" and "porter_test" or just install all packages. After installation the window can be closed and quit Python by entering "`quit()`".

Jupyter was used for this project to run the .ipynb files. If Jupyter is not installed, enter "`sudo pip3 install jupyter`" then "`jupyter notebook`" in Terminal to install and open Jupyter. Python 2 is used for the code, the Python 2 kernel will need to be installed. In Terminal, run "`python2 -m pip install ipykernel`" and "`python2 -m ipykernel install --user`" to install Python 2. Once in Jupyter the .ipynb files can be opened and run easily with the Run button. The code uses data from the file "yelp_labelled.txt" so ensure this file is in the same directory as the .ipynb files so it can be found by the program.