# Yelp Sentiment Analyzer

CSC 664-01

Multimedia Systems

Spring 2020

Stephanie Sechrist

Thomas Sechrist

## Table of Contents

## 1. Introduction

Our Yelp Sentiment Analyzer processes Yelp reviews to predict the star ranking a user assigned a business based on the text in his or her review. It is important, especially during these uncertain times of isolation, because it will help small business owners target what their clientele is looking for when they use their services. By analyzing reviews, it becomes clear to business owners where there is room for improvement and where to focus their efforts. In this report, we will first discuss the project goal and problem it is solving. Next, we will go over a high-level description of the Yelp Sentiment Analyzer to describe how we are attempting to solve the described problem. Then we discuss outside research done that relate to our project. Subsequently, we will go into the specifics of our implementation, including what was done and by whom. Finally, the results of our project will be evaluated.

## 2. Proposed Problem Formulation

The Yelp Sentiment Analyzer takes a review and use text analysis to predict which star ranking a Yelp user gave a business. In doing so, it recognizes that certain phrases and word combinations lend themselves to being positive or negative. Business owners can use this tool to get a better sense of their customers' opinions and what leads them to choose their ratings.

For some businesses, just knowing an overall Yelp rating is not enough information to make informed business decisions. To maintain a customer base, it is important to know why clients value the business and what keeps them coming. Furthermore, to make improvements, it is useful to know exactly what a negative review is based on. If there are many reviews in the same star ranking category with common phrases or words, it is a clear area that needs to be addressed. However, reading many Yelp reviews, especially for a larger establishment, can be time consuming. By using our Yelp Sentiment Analyzer, the user can see which phrases or words are most common to each ranking, allowing them to choose which area of the business to focus on.

## 3. High-Level Description

To build the Yelp Sentiment Analyzer, we used the Yelp Dataset to train our AI using a sample size of 100,000 reviews. Our approach to be able to predict ratings based on the text is to assign reviews a score; depending on the range that the score falls in, we can predict the star rating associated with it.

The first step in the text analysis is to use a text parser to remove whitespace and punctuation. Next, we pre-process the reviews with a stop-list and stemming. Then, we create four bigrams, each corresponding to 5, 4, 2, and 1-star reviews. We do not need to consider 3-star reviews, because they are considered neutral. Now, using these bigrams, we create a weighted vector in the form of a list that contains all pairs of terms from the four bigrams. To do this, each bigram is loaded into the weighted vector, pair by pair, along with a value. Each bigram is associated with a modifier, as shown below.

| Star Rating | 5 | 4 | 2 | 1 |
|---|---|---|---|---|
| **Modifier** | 2 | 1 | -1 | -2 |

The value being added to the weighted vector is the product of the modifier and the number of times the pair occurs in the bigram. If the bigram does not exist in the vector, it is added along with a value; if it does exist, it updates the value associated with that pair by adding to it. Now that we have our trained weighted vector, we save it to be used against new reviews.

When feeding in a new review to predicted, the review is pre-processed in the same manner as the training data and turned into a bigram. Each pair of words in this bigram is searched for in the weighted vector. If the pair is not in the vector, nothing happens; if it is, then there is a value associated with that pair in the weighted vector. We take that value, take its fifth root, and multiply it by how many times the term showed up in the new review. This is done for every pair found in the new review, keeping a running total. The reason we take the fifth root is so that terms that show up often do not have as extreme of an effect. For example, if “great service” has a very large value associated with it, a negative review that has this pair in it would be predicted to have a higher star rating, even though “terrible review” influenced their review more so than the service. It would lead to many false results if one pair of words can greatly influence the score.

Now, with a total found for the review, we need to normalize it so that longer reviews do not always have a higher score than shorter reviews. To do that, the total is divided by the square root of the total number of pairs in the new review. Otherwise, a long one-star review would have a higher score than a short 5-star review. Finally, depending on what range the score falls in, the star rating associated with the review can be predicted. To determine the ranges, we tested 2,000 reviews from each star category using our weighted vector and found the average, maximum, and minimum scores.

| **Score Range** | >= 7.25 | (7.25, 3.6] | (3.6, -2.25] | (2.25, -6.2] | < -6.2 |
|---|---|---|---|---|---|
| **Predicted Rating** | 5 | 4 | 3 | 2 | 1 |

## 4. Prior Work in the Area

When researching ways to approach the problem outlined in section two, our team mostly tried to employ some techniques learned in the text analysis portion of this class, such as stop-lists, stemming, and term frequency tables. However, we did find many projects that used sentiment analysis to categorize text. In general, it was used to find words or phrases that tended to be positive or negative. For example, data scientist David Robinson used sentiment analysis to classify Yelp reviews using R and was able associate words with a sentiment score in [2]. This is similar to what we tried to do, where each pair of words contributes to the review's overall score, allowing us to predict the star rating.

Another project, by Suzana Iacob, used sentiment analysis to analyze a specific business's Yelp rating. In [1], she took out words that leaned heavily towards positive or negative, such as "great" or "worst", and she was able to find specifically what words made people decide to give a restaurant a one-star rating. In other words, these words were used so often that they were overshadowing other important terms. This approach is interesting, but it would not necessarily work for us, because we train our weighted vector using many different businesses, and what works positively for one business might not for another.

The idea of using a weighted vector to assign scores to reviews actually came from the idea of perceptrons in machine learning. We wanted to implement some sort of algorithm that used perceptrons to be able to group our dataset and then assign new reviews to one of those sets. However, the immense dimensionality of the vector made it difficult to implement, and the dataset does not appear to be separable. As we will show in the evaluation section, there is a lot of overlap between star ratings with what we implemented.

Another article that helped us influence our implementation was in regard to stop-lists. Data scientist Gagandeep Sing writes in [3] that it is difficult to employ sentiment analysis on text that has had stop words removed, because it can change the underlying meaning of a sentence. It is for this reason that we did not use a stop word library, such as that found in the Natural Language Toolkit but instead opted for a much shorter list. Our hope, especially since we are using the bigram approach and not bag-of-words, was to keep as much meaning when processing the text for analysis. However, as we will see in the evaluation section, some words should have been included in that list.

## 5. Description of Our Work

### 5.1 Overview of System Description

When looking at our code, there is a decent amount of it commented out, and that is because there were two stages of development that our project went through. First, we created the dataset, bigrams, and weighted vector. Second, we saved the vector and no longer needed the code used to create it. The code left that is not commented out is what we use after saving the weighted vector to predict new reviews' ratings.

### 5.2 Specific Technical Details for Each Part of the System

**Section: Main.py** (Thomas Sechrist)

This file is where we loaded in the JSON data from the Yelp Dataset and cut it up into smaller, more manageable chunks based on star rating, because the file was quite large. These extracted reviews were then saved to their own JSON file that we could then use to extract information. In Main, we created the four bigrams using the TextParser and TermFrequencyTable classes. Then each bigram is loaded into the instance of the weighted vector, which is saved to its own JSON file. Once we had this vector, we commented out all of that code, but we want to keep the code there, because we hope to update the weighted vector to be more accurate in the future. Given more time, we would also like to move the creation of these bigrams and weighted vector into another class. As of now, Main calls and displays the GUI.

**Section: Stem.py** (Stephanie Sechrist)

This is one of the only sections where we utilized the Natural Language Toolkit. We originally wanted to implement our own stemmer but quickly determined that our time would be better spent toward other aspects of our project, so we used this library. It takes a list of words and returns a list of stemmed words in the same order, without omitting repeated terms. It is called from StopList and TextParser.

**Section: StopList.py** (Stephanie Sechrist)

This class simply holds a list of words that will be omitted from text. There are two lists here, because as described in section 4, we found research saying that in sentiment analysis, it is not always beneficial to remove stop words. As we will describe when explaining TermFrequencyTable, the way we build the bigram made it even more of a hindrance to omit some words that we were previously removing. As a result, the new stop-list (list2) has more general terms that are not specific to Yelp reviews.

**Section: TextParser.py** (Stephanie Sechrist, Thomas Sechrist)

For text analysis, we needed to clean up the text to make it easier to process. Main calls this class for each review in a given JSON file. formatText was the original text parser we used when we were using single word term frequency tables, but we created formatTextNGram to help implement bigrams instead. In formatTextNGram, we first use Python's split() function to split up the review being parsed into a list, using space as a delimiter. We then feed that list into Stem to get the list trimmed back to root words; this gives us stemmedList. We then go through each pair of words in stemmedList; make them lowercase; remove all symbols and whitespace in each word in the pair using regular expressions; and then assign it to newEntry. We then go through list2 from StopList and check every word in each pair; if either of them is in list2, they entire pair is considered invalid and not added to the bigram. If the pair of words passes this test, then it is appended to listText (the bigram for the current review being parsed) with a counter.

In addition, there are functions to sort the results, which we used mostly during development to make sure things were working. There is also a getter.

**Section: SampleReviews.py** (Stephanie Sechrist, Thomas Sechrist)

This file consists of our original hand-picked Yelp reviews that we used to make sure our pre-processing techniques and term frequency table creation were working fine.

**Section: gui.py** (Stephanie Sechrist)

This GUI was created as a simple interface for the user to test our application. It is called from Main. It has its own instance of WeightedVector so that it can take the text inputted from the user and make a prediction of the rating. When the user clicks the "Click here to predict the star rating" button, it calls the predict() function from WeightedVector. When the user clicks the "Start over" button, the text field is cleared and no prediction shows. The GUI is pretty simple, because it was my (Stephanie) first time programming a gui in Python.

**Section: TermFrequencyTable.py** (Thomas Sechrist)

In this class, we use formatTable, called from Main, to format the bigram. When using TextParser, the resulting bigrams tft1, tft2, tft4, and tft5, contain all pairs of words and their number of occurrences one review after another. In other words, there will be repeated pairs of words. In formatTable in TermFrequencyTable, all of the redundancies are removed so we have the total number of occurrences of a pair of words for all reviews in a specific star category. We then sort the result alphabetically. The rest of the functions in this class were used primarily when we were testing results and wanted to see the bigrams.

**Section: WeightedVector.py** (Thomas Sechrist)

WeightedVector is used in Main and from GUI. updateVector() is called from Main for each bigram (tft1, tft2, tft4 ,tft5) and takes the bigram and the modifier (see section 3) as parameters. This is where the weighted vector is created as compareVector, adding keys (the pairs of words) and the corresponding values. To reiterate, the value is determined by multiplying the modifier associated with the star rating by the number of times the pair of words occurs in the bigram.

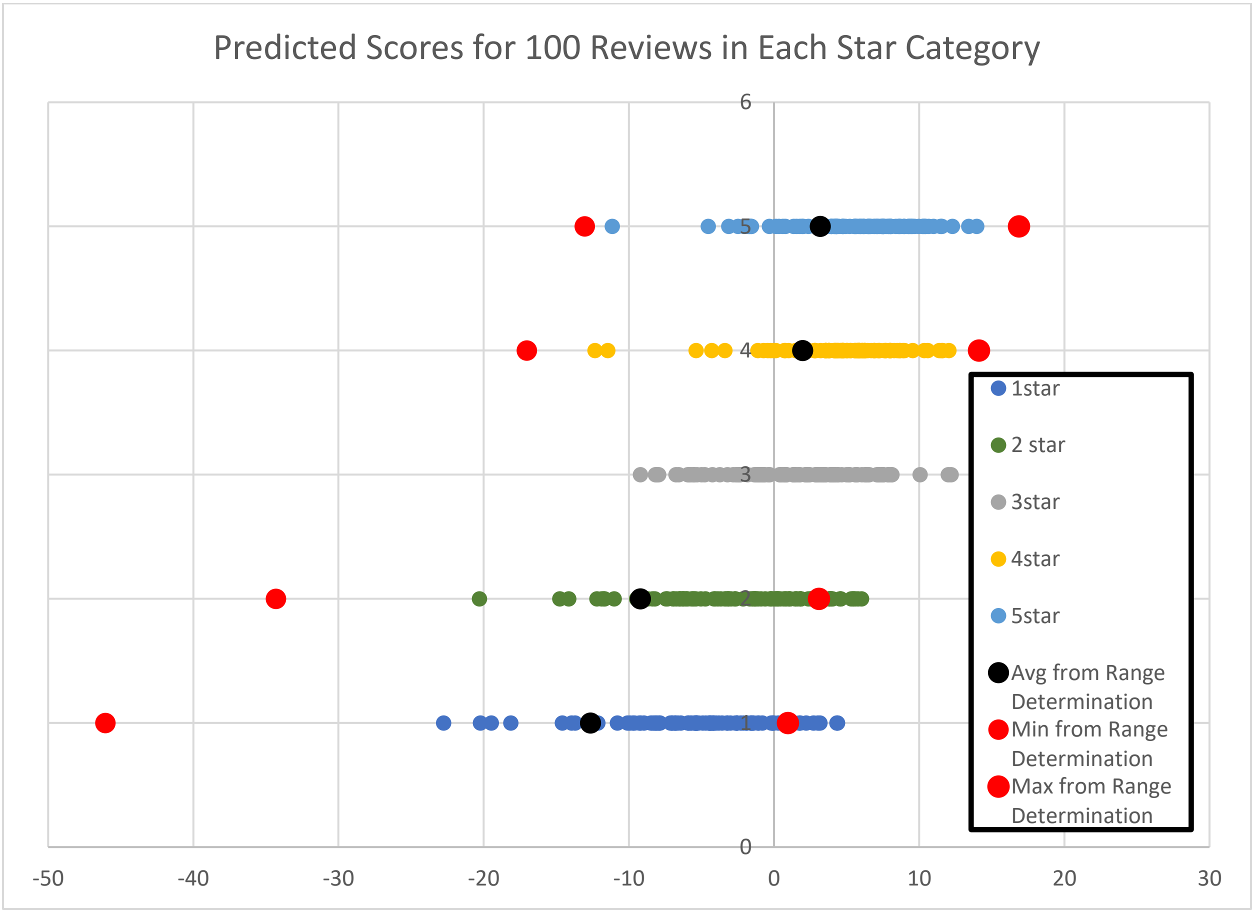
The eval() function takes  a review and assigns it a score. First, it uses to TextParser to pre-process the review and turn it into its own bigram. Then it looks at each pair of words in the review and looks for it in compareVector. If it is not found, nothing happens. If it is, it takes the value associated with the word pair from compareVector, takes the fifth root, multiplies it by the number of times the pair occurs in the review, and adds the result to a running total. Now, this running total needs to be normalized, and we do so by dividing it by the square root of the total number of pairs in the review. Finally, we added 4.7 to the scores. We ended up adding this later during our experimental evaluations to make the test results center around zero.

## 6. Experimental Evaluations

To test our approach, we fed our program 100 reviews from each star category with the following results:

| Actual Rating | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Avg Predicted Rating | 1.98 | 2.47 | 3.19 | 3.85 | 4.04 |
| % Correct | 35% | 21% | 43% | 50% | 36% |

As you can see, the results are not ideal. Upon further investigation, it looked like some common and not meaningful phrases were skewing the results, such as “and they.” Unfortunately, this is an issue with our stop-list, so given more time, we would have to go to the beginning and rebuild our dataset, bigrams, and weighted vectors.



Above is a graphical representation of our test results, showing the scores predicted for each review. As you can see, there is a lot of overlap between 5 and 4, as well as 1- and 2-star reviews in terms of the predicted score. The red dots show what the minimum and maximum scores were for each star rating when developing the score range. The black dot shows the average. Because the 4- and 5-star dots are so close together, it is especially difficult for our program to accurately predict these ratings. One idea we had to improve our results would be to try to shift the average for the 5- and 1-star reviews more positive or negative respectively by changing the modifiers.

## 7. Conclusions

Overall, the Yelp Sentiment Analyzer is on the right track to be able to accurately predict results. Out of the 500 reviews we used in testing, 185 reviews were correctly predicted, which is 37%. If we were to assign reviews a random rating from 1 to 5, there is a 20% the prediction would be correct. Our results are higher than that, so for our first implementation, we are content with the project overall. There is definitely room for improvement, so in later versions of this application, we plan to update the stop-list to include more terms that we see greatly affect results even though they do not hold meaning. We also think that using trigrams would potentially cut back on the amount of heavily weighted phrases. Furthermore, we would like to work more on adjusting the modifiers that help us build the weighted vector to hopefully give us more separable results.

## 8. References

[1] Iacob, Suzana. "Sentiment Analysis of the Yelp Reviews Data." Kaggle, 7 Mar. 2019, www.kaggle.com/suzanaiacob/sentiment-analysis-of-the-yelp-reviews-data/comments.

[2] Robinson, David. "Does Sentiment Analysis Work? A Tidy Analysis of Yelp Reviews." Variance Explained, 21 July 2016, varianceexplained.org/r/yelp-sentiment/.

[3] Singh, Gagandeep. "Why You Should Avoid Removing STOPWORDS." Medium, Towards Data Science, 24 June 2019, towardsdatascience.com/why-you-should-avoid-removing-stopwords-aa7a353d2a52.