Yelp Review Sentiment Analysis

Zihao Xu, Yu Xuan Hong, James Ren

December 16, 2017

Abstract

In this paper, we explored and performed sentiment analysis on reviews obtained from the Yelp Data Challenge. We explored temporal trends of review sentiment as well as interesting interactions between review tags. We also build a sentiment predictor for reviews solely based on text analysis. The resulting interactive module allows us to assess the sentiment of a unlabeled piece of review text with a confidence score.

Contents

1 Introduction

Yelp is an American multinational corporation headquartered in San Francisco, California. It was founded in 2004, and dedicated to help people find great local businesses like dentists, hair stylists and restaurants. The company allows users to give star ratings and write reviews for businesses. More than 142 millions of reviews have been written by Yelpers by the end of Q3 2017. Among these reviews, myriads of hidden information can be extracted, and these information can be beneficial to business owners as well as customers. By studying these reviews, owners can be more aware of things that they need to improve, and customers will be able to find better matches for their own tastes. How to extract the most important information, however, has remained a big problem for a long time. This project specifically focuses on building a model that can extract useful features from millions of reviews. Processing these reviews using sentiment analysis, finding the most featured words and predicting the proper ratings solely based on review text.

2 Problem & Motivation

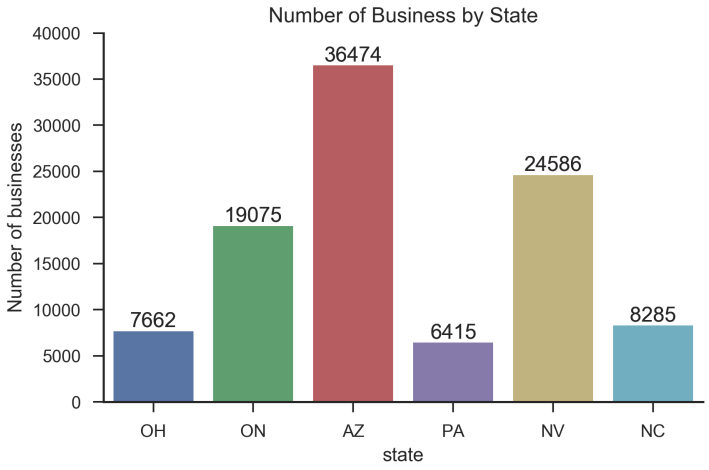
When we go out with friends and dine at restaurants with high ratings on Yelp, we sometimes feel that these restaurants are overrated by others. This happens because different people have different tastes and standards for restaurants. If there is a mismatch between our tastes and the reasons behind the high rating of a particular restaurant, we may experience such "deception of high rating". Luckily, Yelp not only allows customers to rate a restaurant on a scale of one to five stars, but also encourages customers to write reviews about their dining experiences. We believe that, compared to star ratings, the text-rich reviews are more indicative of restaurants' properties and user preferences. If we can extract the most important keywords from customers' reviews, we can not only identify hidden properties of businesses, but also accurately depict individual customer's profile. However, reading all the reviews one by one can be tedious and time-consuming. We hope to build a model that can help people analyze each review, extract the most important information from these reviews, and predict the rating of a review solely based on its text. During the process of doing so, we also hope to find some hidden trends, either about how people's sentiments towards the same word have changed, or how people's tastes have evolved over time.

3 Data Description

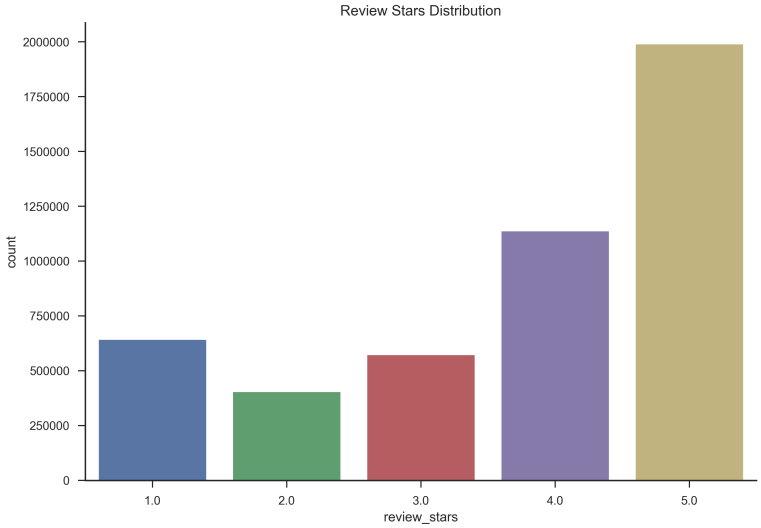
The dataset used for this project is readily available from Yelp Challenge at this link. The dataset contains information about businesses as well as users in 12 metropolitan areas across four countries. There are a total of 6 datasets, among which we will mainly work with `review.json`, `business.json` and `user.json` for our model construction. In all of our files, `review.json` is the largest one, containing over 4.7 millions reviews on Yelp from 2004 to 2017, and has a size of 3.82 GB. Each review is accompanied by its business id, user id, date, stars (on a scale of 1 to 5), review id and its original text. All ids in our files(Business id, user id,review id) are represented by randomly assigned combinations of numbers and letters. This is for protecting the privacy of Yelpers while serving as unique identifiers.

Before we start looking into our data processing, we first have to define two terms that will be used repetitively in our project, and are essential to our project. Throughout the whole project, we define positive reviews to be reviews with star ratings higher than 3, and negative reviews to be reviews with star ratings lower than 3. We do not put reviews with 3 stars to neither categories, because these reviews seem to be ambiguous, and can be reasoned to belong to either side.

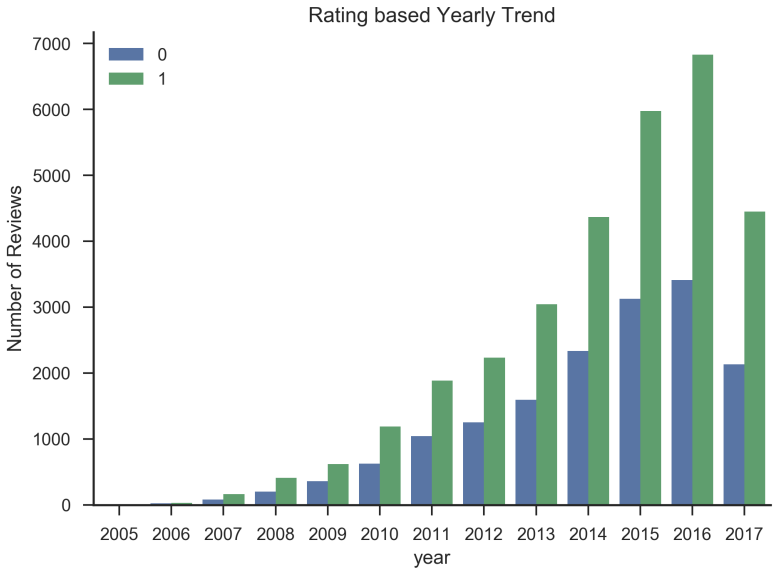
To perform our analysis, we first converted all `.json` files to `.csv` files, merging `review.csv` with `business.csv` and `users.csv` respectively by using each user and business' unique id. Then, we filtered through some unnecessary information, leaving only data from top 6 states with the most reviews. Among which we further filtered our dataset, and only kept business with 10 or more reviews and users who has written 5 or more reviews. This process cut the total number of reviews down to 4265614. The following graph will give you an idea about how these reviews are distributed based on states, review stars, and time stamps.



From this graph, we notice that Arizona has the most reviews, which was surprising to us at first considering the fact that Arizona is not often considered as one of the most featured state in US. From this graph, we suspect that our data set may be slightly biased due to the fact that it is only a small portion from Yelp.



Now if we look at the distribution for review stars, we see an upward trend from 2 stars to 5 stars. This signifies that people tend not do give natural ratings. They usually rate businesses when having relatively strong emotions, as there are more 1 star, 4 stars, and 5 stars reviews than 2 stars and 3 stars reviews. It is also interesting to see that there are much more positive reviews than negative reviews. This uneven distribution can potentially be a challenge for us when training our model.



The last graph records the number of reviews written by Yelpers each year. We observe that, while the ratio of positive and negative reviews stays relatively stable, the total number of reviews has been increasing monotonically each year. It is important to point out that the number of reviews in 2017 is lower than previous years because we only had statistics for the first half of 2017 by the time we received this dataset.

4 Exploratory Analysis

After processing our dataset, first, we wanted to perform some exploratory analysis. We were curious about things such as the overall distribution and trends of review ratings, relationships between specific words and review ratings, as well as characteristics of all reviews for particular businesses or by particular users. We used tools and packages like Jupyter notebook, Python Pandas, and Seaborn, and they proved to be particularly helpful for these types of analysis. We were able to generate many plots, and interpret some of the results to our queries.

In the following subsections, we will describe the processes of converting the data into desired formats, pruning the dataset, testing our review analysis on sample data, performing business/user specific analysis, and finally, scaling our analysis over the entire 5 GB of data.

4.1 Initial Processing and Pruning

As mentioned in the previous section, our initial datasets were in `.json` format. Because `.json` format does not work well with Data Wrangling packages like `pandas` and `seaborn` we needed to convert the data into `.csv` format.

We are now left with three `.csv` files: `review.csv`, `user.csv` and *business_s*`.csv` `review.csv` contains information for each review: each review has a unique review id, with columns including its corresponding user id, business id, review text, rating, date, and tag counts for "cool", "funny", and "useful". `user.csv` contains information for each user, including the user's unique user id, average stars given, counts for various tags, list of friends, and join date. *business_s*`.csv` contains information for each business, including the business' unique business id, its taxonomy list, attribute list, location, price range, business hours, accepted payment forms, etc.

Since our analysis was mainly review based, information in `user.csv` that was useful to us were user id and average stars. In *business_s*`.csv`, we reformatted its attribute list and created new columns denoting price information and whether or not the business accepts credit payment.

Finally, for different purposes of our analysis, we created data frames that merged business information and review information on business id, as well as user information and review information on user id.

An initial probe into `business.csv` reveals that our businesses are mainly represented in about 15 states. We decided to consider only those businesses that are located in the 6 most represented states: OH, NC, ON, AZ, PA, and NV, which in total counted for about 86.57% of the entire data set. Figure 1 shows a distribution of business across these 6 states.

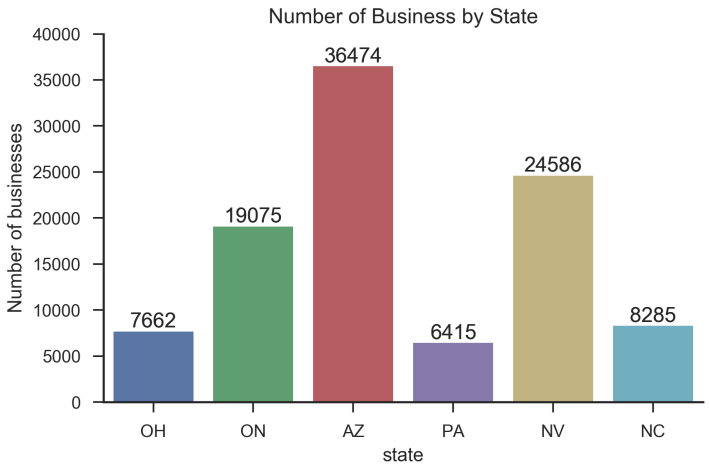


Figure 1: Distribution of businesses across states

We also filtered out business with fewer than 5 reviews, and users with fewer than 2 reviews. We also removed entries with null values on critical columns.

The pruned dataset contains 4,736,897 reviews, 968,039 users and 102,497 businesses.

4.2 Review Analysis

In our review analysis, we mainly explored two aspects: the trend of review sentiment over time, and relationships between review tags and ratings. We first attempted our reviews analysis on a smaller sample. From our reviews dataset, we generated a random of sample of 4737 reviews, which constitutes for about 0.1% of the entire dataset. We then scaled our code to perform analysis for the entire dataset, which contains over 4,700,000 reviews. The plots included in this writeup are those generated with the entire dataset.

4.2.1 Sentiment Based Trends

We now exploit review text to generate temporal trends of reviews based on sentiment estimation. We first use polarity as a crude estimate of text sentiment for each review. Polarity is formally defined as $\frac{p-n}{p+n}$, where p is the number of positive words, and n is the number of positive words in a corpus. Polarity takes a value between -1 and 1, with -1 representing most negative, and 1 representing most positive. We obtained the polarity of each review text by first stemming the review using NLTK (further discussed in subsection 5.2.3) and then applying the built-in polarity function of Textblob[1] to the stemmed corpus.

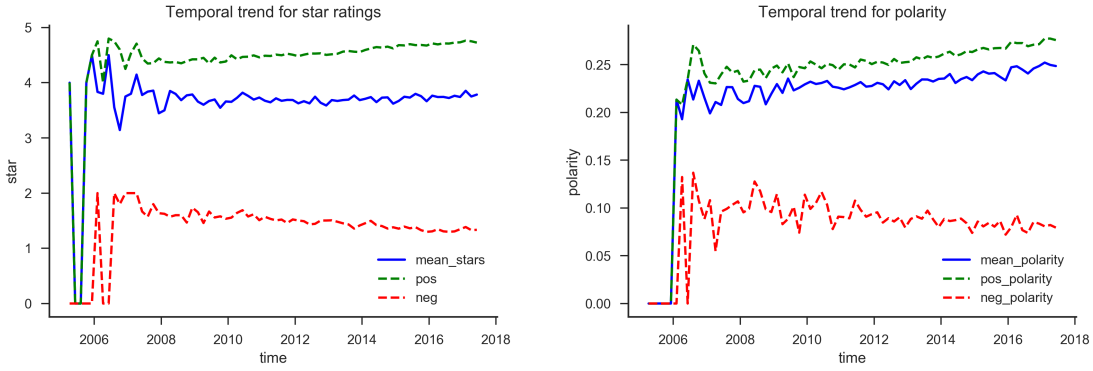
We then categorized reviews based on their ratings. We consider reviews with 4 or 5 stars to be positive reviews, and those with 1 or 2 stars to be negative reviews.

Figure[2a] shows the temporal trend of star ratings of positive and negative reviews. Figure[2b] shows the he temporal trend of polarity of reviews with polarity values above and below the median polarity. In both plots, there is a slight upward trend for "positive" reviews, and a slight downward trend for "negative" reviews. Overall, both ratings and polarity reflect a slight trend towards positivity. This shows that while reviews are overall becoming slightly more positive with time, they are gradually becoming more polarized.

We then considered the distribution of positive and negative reviews over different units of time.

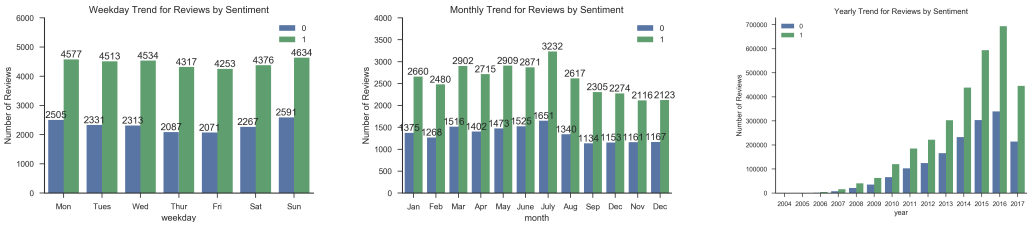
Figures[3a],[3b], and [3c] show the distribution of positive and negative reviews over weekdays, months, and years. Overall, we can see that there are more positive reviews than negative reviews.

[1]For more information, visit official documentation of Textblob.polarity



(a) Rating trend (b) Polarity trend

Figure 2: Sentiment Trend over Time



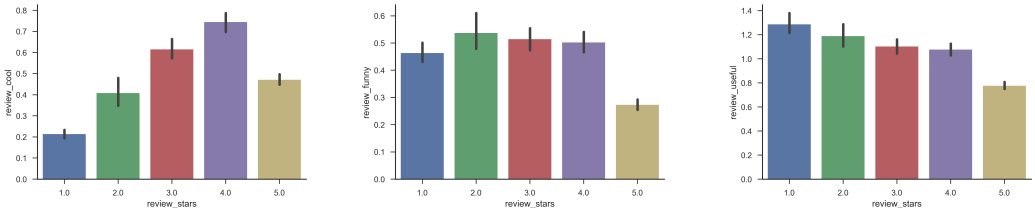
(a) Weekly trend (b) Monthly trend (c) Yearly trend

Figure 3: Rating Distribution over Time

Reviews seem to be evenly distributed over different days of the week and months of the year; they have been increasing in numbers by year (the slight dip in 2017 is because the data is recorded until July of 2017.)

4.2.2 Review Tag Analysis

In this part of the analysis, we consider the tags of reviews. For each review, there are three possible tags that other users can give votes for: “cool”, “funny”, and “useful”. We were curious about the relationship between these tags and review ratings, as well as correlations between counts of different tags.



(a) Average “Cool” (b) Average “Funny” (c) Average “Funny”

Figure 4: Average Tag per Review

Figures[4a], [4b], and [4c] show the distribution of average votes for a tag per review over ratings. Some interesting observations include that the average “usefulness" of reviews increases as their ratings go down, and that 4 star reviews are on average the “coolest”.

To explore the relationships between tag words and ratings, we plotted the following heat maps (Figures[5],[6], and [7]). Each data point represents instances of two tags both occurring at their respective votes, and the color of the data point reflects the average rating of all such occurrences.

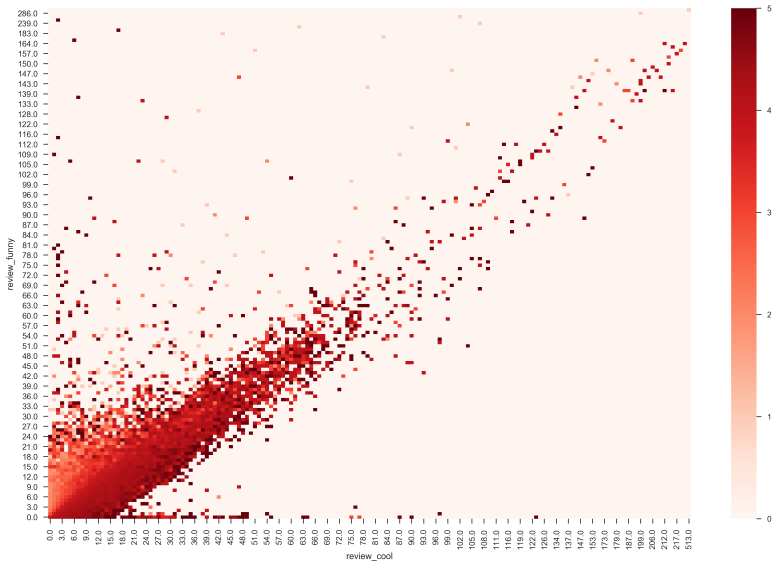


Figure 5: “Funny” - “Cool” Heat Map

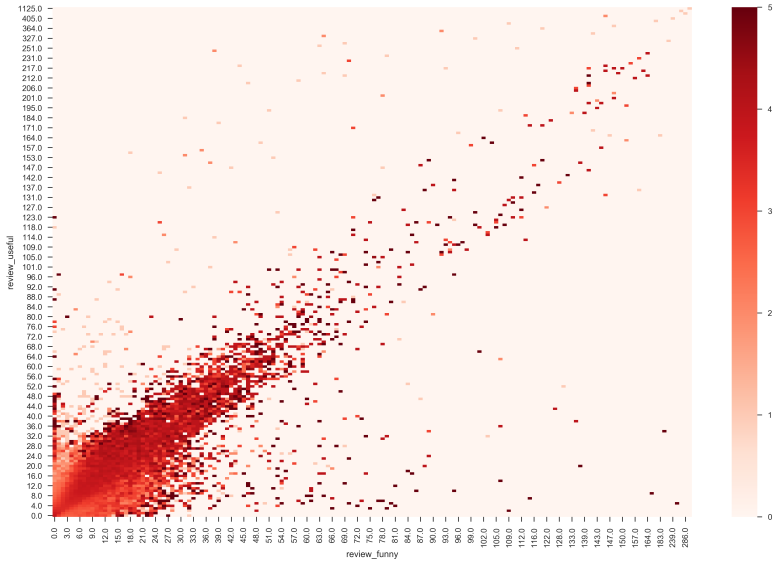


Figure 6: “Useful” - “Funny” Heat Map

If we taker a closer look at Figure[7], we notice some very interesting phenomena. There is a significant diagonal trend, spanning from the lower left corner to the top right corner, depicting those data points that have roughly the same percentile of “usefulness” and “coolness”. The darker color shows that these reviews often have high ratings. On the other hand, above this data trend towards the left side, there is a small, non-linear patch of lighter colored data points. These depict reviews that are “useful” but not as “cool”. Their lighter color imply that they usually have lower ratings. These data trends are quite consistent with what we expect with the usage of tags.

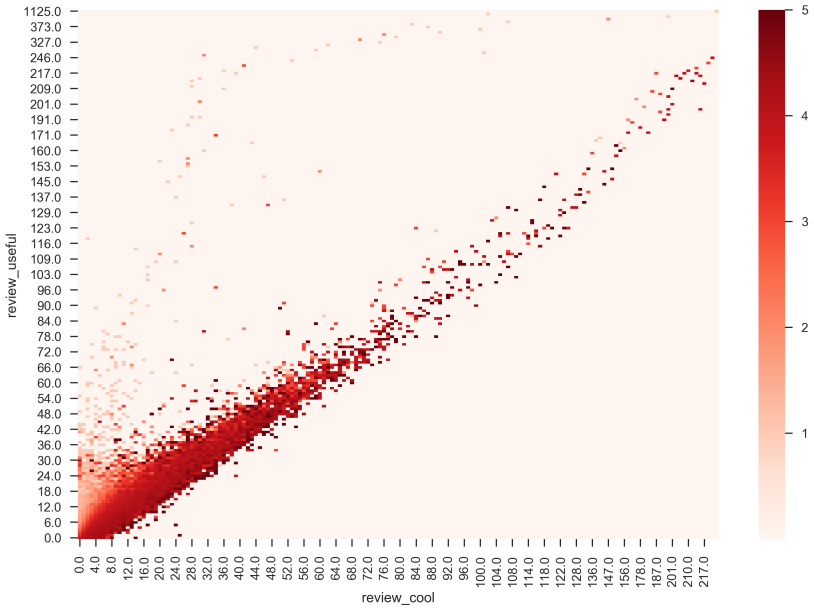


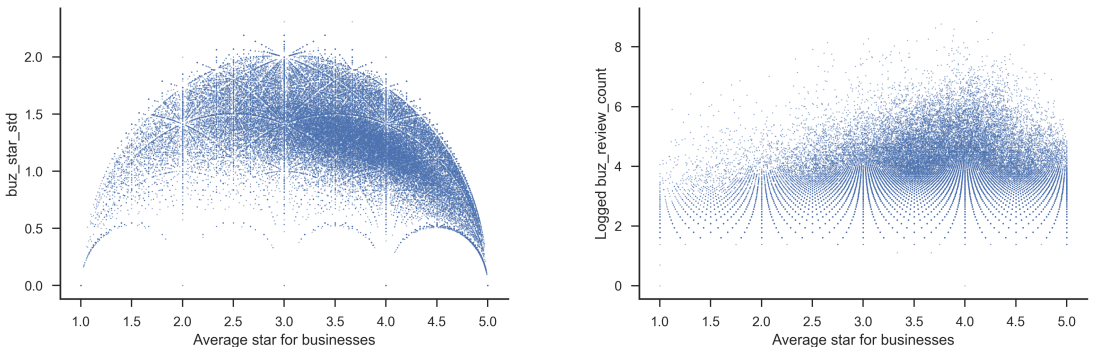
Figure 7: "Useful" - "Cool" Heat Map

4.3 Business and User Analysis

Our initial goals for this section of analysis was to provide a categorization for businesses and users based on the reviews they are associated with.

For each business, we plotted its average review ratings against the polarity and subjectivity of an aggregation of its reviews (both generated by Textblob).

We also plotted its average review ratings against the standard deviation of its reviews. Finally, we plotted its average review ratings against the number of reviews it received.



(a) Average - Standard Deviation

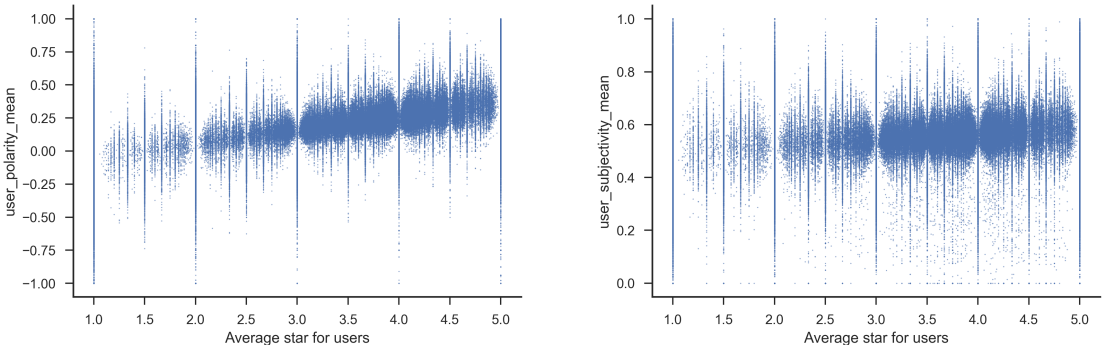
(b) Average - Count

Figure 9: Business Rating Statistics

We made similar plots for each user, shown in Figure 10 and 11.

5 Sentiment Analysis

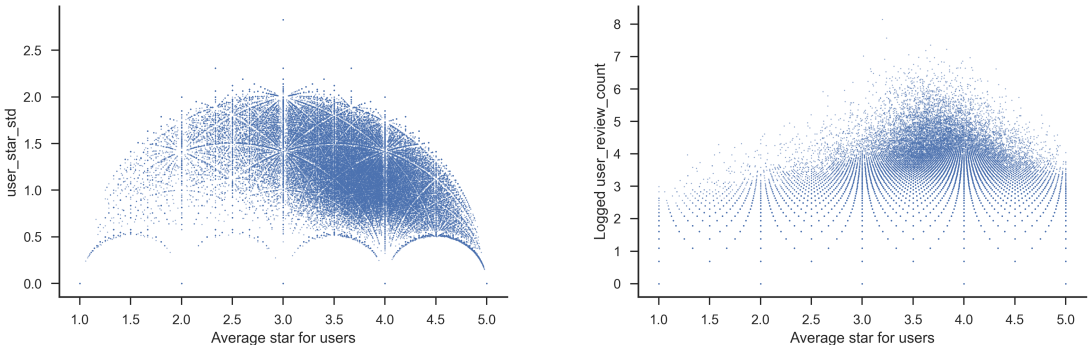
After exploring the data, we now move on to the the discussion of sentiment analysis based on review text, the major focus of our project. As mentioned in Section 1, we would like to understand the reasons behind users' positive/negative ratings, and be able to assess the sentiment of the reviews solely based on the text itself. We believe that review texts are more direct representation of users' actual experience at the business and thus should contained much more information than star ratings. Therefore, in this section, we will first talk about text pre-processing; then we will



(a) Rating - Polarity

(b) Rating - Subjectivity

Figure 10: User Rating - Sentiment Trend



(a) Average - Standard Deviation

(b) Average - Count

Figure 11: User Rating Statistics

move on to several machine learning methods used in our analysis; finally we will talk about a voting classifier and an interactive sentiment predictor.

5.1 Data Preparation

The data used in this analysis comes from the pre-processed review.csv mentioned in Section 3. The only useful columns are *text* - the actual review text that users give to the business in string format - and $review_stars$ - the star rating associated with the review. A third column named *pos* is created based on $review_stars$ as follows: reviews with 1 or 2 stars are assigned 0, indicating negative sentiment; reviews with 4 or 5 stars are assigned 1, indicating positive sentiment; reviews with 3 stars are temporarily discarded as it is hard to categorize 3-star reviews as either positive or negative. After dropping the 3-star reviews, we ended up with 4265614 valid reviews tagged with 0 or 1 in the *pos* column.

5.2 Text Pre-processing

Aside from assigning sentiment, we also preformed several necessary pre-processing on review text, including tokenizing, removing stop words and stemming. We explored lemmentization[2] of individual words as an alternative to stemming, but we have discovered that this form of pre-processing leads to inferior prediction accuracy than that of stemming.

[2]For a more complete discussion on this topic, please refer to this link.

5.2.1 Tokenization

Tokenizing refers to the act of separating a sentence into individual word tokens. It is the very first step towards processing individual words in a piece of text. In our analysis, we used the default *word_tokenizer* in the *nltk* package under Python. As an example, the following sentence is passed to the *word_tokenizer*:

The python programmer named pythoner is pythoning a game pythonly

The result returned by the *word_tokenizer* is as follows:

['*The*', '*python*', '*programmer*', '*named*', '*pythoner*', '*is*', '*pythoning*', '*a*', '*game*', '*pythonly*']

5.2.2 Remove Stop Words

After tokenization, we need to remove the common stop words in English. Stop words are words that appear extremely common but do not convey much meaning nor sentiment of bodies of text (JACOB, May). We used the bag of stop words[3] gathered by the *nltk* package and eliminated the words that falls into this bag from each piece of text.

Continuing with the example above, we filter the tokenized list of word by the bag of stop word and obtain the following filtered list:

['*the*', '*python*', '*programmer*', '*named*', '*pythoner*', '*pythoning*', '*game*', '*pythonly*']

Observe that words like "is" and "a" are removed from the list. whereas words that convey meaning are kept.

5.2.3 Porter Stemmer

Then we stemmed the individual words in each piece of text using Porter Stemmer (C.J. van Rijsbergen and Porter, 1980). Stemming refers to "the process of reducing inflected (or sometimes derived) words to their word stem, base or root form generally a written word form". In particular, Porter Stemmer is the de facto algorithm developed in 1980, and it was widely used to stem English words. In our analysis, we used the *nltk* implementation of Porter Stemmer to bring individual words from text to its stem form. Note that during the processing steps, words with the same stem are assumed to convey roughly the same meaning, and therefore are weighted equally in sentiment analysis.

After passing the above list of words to a Porter Stemmer, the list returned is in the following form:

['*the*', '*python*', '*programm*', '*name*', '*python*', '*python*', '*game*', '*pythonli*']

Notice that all python-related words, except "pythonly", are being turned into its stemmed form "python". The single exception of the adverb might be an example of under stemming (Jivani, 2011).

5.3 Feature Selection

Besides pre-processing, another important area that we explored is feature selection for sentiment prediction. Feature selection for sentiment prediction is a popular topic with many on-going researches (Duric and Song, 2012) (Koncz and Paralic, 2011). Popular methods for feature selection include information gain (Koncz and Paralic, 2011) and HMM-LDA (Duric and Song, 2012). For our purposes, we use a rather simplistic but practical set of features: individual words, bi-grams and capitalized words.

Our hypothesis is that, individual words are able to capture most of the information within the sentence; bi-grams helps with negations like "not like" or "not recommend"; and capitalized words like "ZERO" and "BEST" indicate very strong sentiments. To testify, we have identified a bag of potential words using the Naive Bayes Classifier from the *nltk* package (discussed further in Section 5.5.1) and plotted the distribution of review stars for reviews containing these key phrases (in red) and overall distribution of review stars (in blue):

[3] This bag of word is loaded using *stopwords.words('english')*. For a complete list, visit this link.

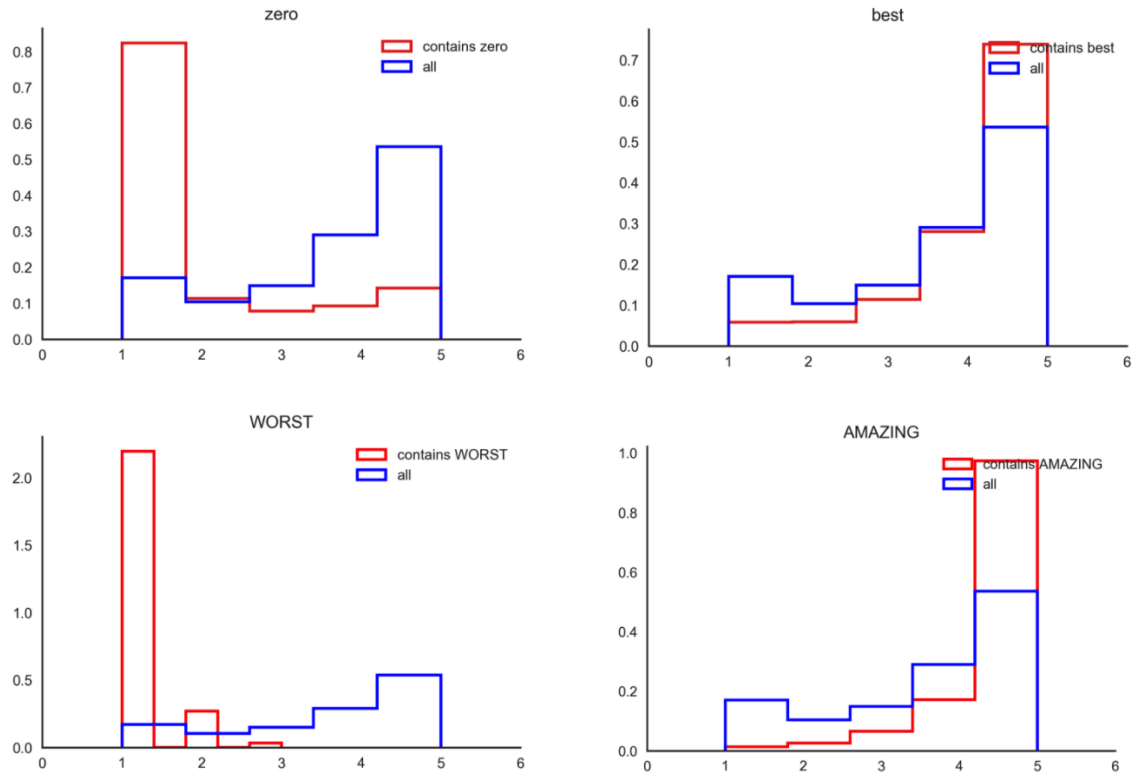


Figure 12: Distribution of review stars for review containing emotional words

Notice that the distributions in red are starkly different from the overall star distribution for all the selected words, indicating that these words indeed have predictive power in determining the sentiment of texts.

5.4 Spare Matrix Representation

The final step towards model training is transforming the body of text into a spare matrix representation. To this end, we have explored two different ways of transformations, Count Vectorizer and Tf-idf Vectorizer.

The idea behind Count Vectorizer is quite simple: we first identify all the unique words that are present in all of our 4 million review texts. Then we create a matrix whose columns are individual words and rows represent individual texts. The values within each entry represents number of times the word (or bi-gram) corresponding to the column is present in the text corresponding to the row.

The Tf-idf Vectorizer is more complicated yet more powerful transformation approach. "Tf-idf" stands for "Term-Frequency and Inverse Document-Frequency". The "Term-Frequency" part is exactly the same as raw counts, but the "Inverse Document-Frequency" is what really differentiates Tf-idf from the normal Count Vectorizer. In Tf-idf calculation, the term frequency, the number of times a term occurs in a given document, is multiplied with the idf component[4]:

$$\text{tf-idf(t,d)} = \text{tf(t,d)} \times \text{idf(t)}$$

Figure 13: Tf-idf Calculation

and the idf component is computed as:
where n_d is the total number of documents, and $\text{df}(d,t)$ is the number of documents that contain term t. The resulting tf-idf vectors are then normalized by the Euclidean norm:

As we can see from the calculations, the idf component favors those word that do not occur too many times across all the bodies of texts, but penalizes the words that occur too many times.

[4]The figures and explanation are referenced from the sklearn documentation on feature extraction.

$$\text{idf}(t) = log\frac{1+n_d}{1+\text{df}(d,t)} + 1$$

Figure 14: Inverse Document-Frequency Calculation

$$v_{norm} = \frac{v}{||v||_2} = \frac{v}{\sqrt{{v_1}^2+{v_2}^2+\cdots+{v_n}^2}}$$

Figure 15: Euclidean Normalization

The overall effect of this transformation is that more weights are transferred from terms that are more ubiquitous across texts to those that are more unique to specific texts.

One other optimization step we could do at this point is to specify the range for number of times that a word appears in the collection of texts: we do not want words that appear in every piece of text, as these words might be stop words that were not filtered out at the pre-processing step, while words that appear too few times might be actuals features that we are looking for. We have picked $min\_df = 5$ and $max\_df = 0.95$ to indicate that a word need to appear in at least 5 distinct pieces of text but should not appear in more that 95% of all the texts (roughly 3.8 million) to be considered a feature. This step has a similar effect with the Tf-idf transformation and it is able to drastically reduce the number of features included, which would consequently lessen our computational burden. Finally, the total number of features, including bi-grams and capitalized words, becomes 444315.

5.5 Model Training and Evaluation

5.5.1 Training

For sentiment predictions, we have tested a few machine learning algorithms that can make binary (1 for positive and 0 for negative) classifications using different sets of features. We have explored: 1. Naive Bayes algorithm from the *nltk* package, trained on sparse matrix built by only individual words as features; 2. several more advanced algorithms, including MNB, BernoulliNB, LogisticRegression, LinearSVC, PolySVC, RadialSVC, NuSVC and SGDClassifier, from the *sklearn* package, trained on sparse matrix including only featured individual words; 3. advanced algorithms, trained on sparse matrix including a combination of individual words, bi-grams and capitalized words. For each model, we divided the data into training and testing set, and then fed the transformed spare matrix representation of the training set to each algorithm, performing an evaluation on each algorithm using the transformed (into the same sparse matrix) testing data.

5.5.2 Evaluation

Before we start discussing about the results, it is worth pointing out that we have observed a skewed distribution in review stars from our data set: there are 75% of positive reviews among 4 million reviews used. Therefore, we should rule out any classifier that produce less than 75% test accuracy because it does not outperform the strategy of predicting positive for every piece of review text.

Naive Bayes Algorithm
For the algorithms we have explored, the Naive Bayes Classifier performed rather poorly, yielding roughly 46% test accuracy, But we were able to gather a set of so-called "Most Informative Features" from the algorithm:
Observe that although the algorithm did poorly in predicting the sentiment of texts, it did produce a bag of informative words that are indicative of strong (mostly negative) sentiments.

More Advanced Algorithms
Then, we fed the spare matrix into some more advanced algorithms, including MNB, BernoulliNB, LogisticRegression, LinearSVC, PolySVC, RadialSVC, NuSVC and SGDClassifier. We evaluated the classifiers using test accuracy and auc score (the area under ROC curve). The results for the three top-performing algorithms are as follows:

```
Original Naive Bayes Algo accuracy percent: 46.200980392156865
Most Informative Features
                   refund = True              neg : pos    =     49.9 : 1.0
                     zero = True              neg : pos    =     23.2 : 1.0
                    bewar = True              neg : pos    =     21.7 : 1.0
                     rude = True              neg : pos    =     21.7 : 1.0
                   garbag = True              neg : pos    =     19.8 : 1.0
                     dirt = True              neg : pos    =     19.8 : 1.0
                    ignor = True              neg : pos    =     18.6 : 1.0
                   terribl = True              neg : pos    =     18.5 : 1.0
                 underwhelm = True             neg : pos    =     17.9 : 1.0
                  tasteless = True             neg : pos    =     17.9 : 1.0
                   knowledg = True             pos : neg    =     17.8 : 1.0
                       shut = True             neg : pos    =     16.0 : 1.0
                     intent = True             neg : pos    =     16.0 : 1.0
                 seriously? = True             neg : pos    =     16.0 : 1.0
                      upset = True             neg : pos    =     15.7 : 1.0
```

Figure 16: Most Informative Words from Naive Bayes Algorithm

	Test accuracy	AUC_score
LogisticRegression:	95.48%	95.35%
LinearSVC:	96.45%	94.81%
SGDClassifier:	94.71%	94.77%

Figure 17: Model Evaluation Three Best Algorithms

We can see that the selected algorithms are producing decent results with around 95% of accuracies and auc scores.

Importance of Bi-grams

Besides the model performance, we have also trained the models with and without bi-grams to test our hypothesis that including bi-grams will be able to capture negations like "not good" and "not recommend". The results showed that including bi-grams indeed increased test accuracy by around 2 percentage points, which is pretty significant given the improvement was from 94% to 96%. More interestingly, we tested some sample reviews against the models with and without bi-grams, and results show that the models trained without bi-grams did poorly (gave positive sentiment predictions for negative reviews) because they treated words like "good" and "recommend" as highly positive even though they are negated in the sentence; however, models exposed to bi-grams performed very well in capturing phrases like "not good" and "not recommend" and predicted negative for the given reviews correctly:

```
# The issue with n-grams: do not, not recommend, not good
print(clf.predict(vect.transform(['do not recommend this place',
                                  'this place is not good'])))
```

```
[1 1]
```

Figure 18: Results for Model Trained W/out Bi-grams (1: positive, 0: negative)

```
# NO MORE issue with n-grams: do not, not recommend, not good
print(clf.predict(vect.transform(['do not recommend this place',
                                  'this place is not good'])))
```

```
[0 0]
```

Figure 19: Results for Model Trained With Bi-grams

5.6 Moving to a Voting Classifier

Another area we explored is building a voting classifier that combined the three top-performing algorithms obtained from the previous section. The rationale behind this method is that a majority-vote based system is able to reduce the variance in between algorithms, and the resulting voting classifier theoretically should outperform individual classifiers as each classifier on itself will perform poorly in certain circumstances due to assumptions we made when building the model.

For this new classifier, we wrote two functions: *predict* and $confidence$, where *predict* combines the results from three classifiers, and output the majority-voted (2/3 or 3/3) class as the final prediction, and $confidence$ is a value between 0 and 1 calculated by the number of classifiers in the majority class divided by three. The confidence score is useful in assessing the agreement or disagreement among classifiers, and it can be indicate how confident we are about our final prediction.

5.7 Interactive Sentiment Prediction using Pickle

Finally, we used the *pickle* module under Python to store all of the trained trained Vectorizers and Classifiers. The *pickle* module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure. "Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "Unpickling" is the inverse operation, whereby a byte stream is converted back into an object hierarchy[5].

With the picked object, we were able to build an interactive module that allows users to import the classifiers and make predictions on given pieces of reviews. The output of this interactive module are both the 0/1 sentiment prediction and the confidence score discussed in the previous section. To test this module, visit our github repo page for more information.

6 Conclusion

We have tried many other different approaches for our project, mostly trying to weight each review based on different factors. For instance, its "usefulness" given by users, the time it is written, its relative positiveness compared with the average rating given by this particular user. Unfortunately, none of these approaches turn out to be helpful to increase the accuracy of our prediction. We also tried to incorporate other advanced models such as Long Short Term Memory Network, but the model turns out to be way slower than our expectation. We were unable to process our whole dataset. There are definitely more things we can work on in the future about this project, including, but not limited to solving problems such as: How to determine two or more sentiments in one sentence? How to predict ratings for reviews that have none or very little featured words? How to categories a sentences that is only understandable within a context? How to differentiate the positiveness of a word under different topics? Up to this point we believe that we have created a decently well performed predictor for star ratings, but if you have any thoughts regarding our project and how to improve it, please let us know!

[5] reference: pickle documentation

References

C.J. van Rijsbergen, S. R. and Porter, M. (1980). New models in probabilistic information retrieval. *London: British Library*, (British Library Research and Development Report, no. 5587).

Duric, A. and Song, F. (2012). Feature selection for sentiment analysis based on content and syntax models. *Decision Support Systems*, Volume 53, Issue 4:Pages 704–711.

JACOB (2010 May). Text classification for sentiment analysis – stopwords and collocations. *Bad Data*.

Jivani, M. A. G. (2011). A comparative study of stemming algorithms. *Anjali Ganesh Jivani et al, Int. J. Comp. Tech. Appl*, Vol 2 (6):1930–1938.

Koncz, P. and Paralic, J. (2011). An approach to feature selection for sentiment analysis. In *Intelligent Engineering Systems (INES), 2011 15th IEEE International Conference on*.