Yelp Sentiment Analysis

A Stat 495 Report
Presented to
Professor Wagaman

Jasmine Horan, Jocelyn Hunyadi, and Alex Liu

December 12, 2018

Table of Contents

List of Tables

List of Figures

Introduction

Whether you're looking for a new burger joint to try for lunch or an upscale Italian restaurant with phenomenal food and atmosphere, Yelp provides all the information you need to make the best selection for you. This review-based search service provides its users with a wealth of information on restaurants in the nearby area with just a few quick keystrokes into the search bar. Are you vegetarian and wondering if a restaurant is a good choice for you? Yelp will tell you if it's liked by vegetarians. Are you looking for a great place for a date? Yelp will let you know if the ambiance of a particular restaurant of interest is just right! Or perhaps you really need to know if you can play a round of pool while waiting for your meal? Yelp has that information, too! In fact, Yelp contains information on over hundreds of business attributes - like parking, ambiance, delivery, hours, and more - that can easily be found and located for any business with a couple clicks.

Beyond the business attributes, price range, and hours of business Yelp provides, an often untapped source of information is the 171 million reviews written by Yelp's many reviewers. For each review, users provide a star rating for the restaurant ranging from 0 to 5 and accompanying commentary that can range from a simple, "The food was great!", to an essay describing the service, food quality, ambiance, and overall dining experience. In all reviews, the emotions, attitudes, and opinions of the particular user are conveyed and ultimately, the culmination of these determines the star rating the user gives the restaurant. Is it possible to quantify and extract this kind of 'data' from each Yelp review? In other words, there exists an ample amount of *sentiment* data within each Yelp review that can be extracted to better inform Yelp users and businesses on what influences the star ratings given by users and received by restaurants.

0.1 Sentiment Analysis

Sentiment analysis is the process of determining the tone of words in order to gain a better understanding of the emotions, attitudes, and opinions behind words and phrases posted online. Also known as opinion mining, sentiment analysis is a field within Natural Language Processing (NLP) that builds systems that try to identify and extract opinions within text. Besides identifying solely the opinion behind text, NLP can extract other attributes of the text, such as the polarity (positive or negative opinion) and the subject.

Sentiment analysis also has many useful applications. With the recent rise of social media, there is now more data that can be analyzed from reviews, forums, blogs, and other social media platforms such as Twitter. Sentiment analysis can be used to gauge customer satisfaction for a product, to determine what new products customers want, and to determine what is frustrating customers or making them unhappy.

For example, at the beginning of the month Netflix (U.S.) announced it was removing the popular show "Friends" from the streaming service come January 1st, 2019. Subscribers were immediately outraged, taking to Twitter, Facebook, Instagram, and other platforms to communicate their dissatisfaction. Within days Netflix announced it would keep "Friends" for another year through the end of 2019, costing them an astonishing $100M. Some type of sentiment analysis was likely used in this situation, and demonstrates the importance and power of customer opinions.

Sentiment analysis can also be used to assess a population's response to current events, such as politics. From restaurants to online stores, many businesses and organizations can benefit from this type of data analysis.

0.2 Objectives

Our specific objectives for this analysis were to wrangle the data provided by Yelp and to perform sentiment analysis to determine if the star rating of a review for a particular restaurant could be predicted by the sentiments present in that review. We also hoped to determine which of the four lexicons from the sentiments dataset (tidytext package) was most accurate in predicting review star rating using classification trees. Our final objective was to identify any possible differences in the sentiment and review star rating relationship between the East and West of the United States and between the United States and Canada using selected, representative cities.

Chapter 1

Data

For their 12th dataset competition, Yelp provided undergraduate and graduate students with a downloadable subset of their review, business, and user data. In its raw form, the data contains $5,996,996$ reviews, $188,593$ businesses, $280,992$ pictures, $1,185,358$ tips (suggestions to the business), $1,518,169$ users, aggregated check-in times for each business, and over 1.4 million business attributes, including parking, delivery, atmosphere, etc. Businesses are from 10 metropolitan areas across two countries. The data is available as a set of six JSON files. For our purposes, only the following two JSON files were utilized:

1. **business.json** - data pertaining to a particular business, including attributes, location, and categorical classification (i.e. Food, Nightlife, Grocery, etc.).

2. **review.json** - data pertaining to a particular review, including the review text, star rating, user id, and business id for which the review was written.

Issues were encountered almost immediately when attempting to load the large **review.json** file into R Studio. The data files provided by Yelp are not pure JSON files, but are rather Newline delimited JSON files. This type of JSON file actually contains multiple JSON files that are each considered their own independent object and therefore, must be handled differently than regular JSON files. Still, even after successfully obtaining the data in R, the large files caused frequent crashes and took an absurd amount of time to run even relatively simple commands. Therefore, we decided to select a small sample of the data to investigate our objectives.

To create a smaller sample of a more reasonable size, we first reduced the initial dataset by selecting only food-related businesses (i.e. restaurants, bars, etc.). Next, we further reduced the dataset by only including the following cities: Cleveland,

Pittsburgh, Toronto, and Phoenix. Cleveland and Phoenix were selected in order to address similarities and differences in reviews, sentiments, and star ratings between the East and West coast of the United States. Pittsburgh and Toronto were similarly selected, but as representatives of Canada and the United States. To be included in our final two datasets, food-related businesses from these selected cities had to have at least 100 reviews. This process and additional data wrangling are outlined below.

1.1 Initial Download and Wrangling

Yelp provides it's competition data as a set of Newline delimited JSON files, which must be initially downloaded from the Yelp competition website (Round 12). Once the data was downloaded, all files were loaded and then stored as RDS files for quicker loading in the future. To reproduce this procedure, please see the instructions contained in the `Yelp_Data_Loadin.Rmd` file. Running this file will load in each JSON file and then save the data as an RDS file.

1.2 Dataset Reduction

The RDS files produced in the section above were utilized for further data wrangling in order to decrease initial loading time. First, the `yelp_business.rds` file contains a column called `categories` which is composed of a string of words that describe that particular business. A business was kept in our sample if its category string contained one or more of the following words: Food, Restaurant, Bars, and Nightlife. The resulting business file contained $76,725$ businesses.

Next, we further reduced the amount of data in our sample by filtering for our selected cities. Two separate samples were produced. The first contained $7,215$ businesses from Cleveland and Phoenix for our East-West comparison. The second contained $13,435$ businesses from Pittsburgh and Toronto. Using the `business_id` column, the review data was also filtered to include only the $7,215$ and $13,435$ businesses, respectively.

In the final step to obtain a sample of suitable size, the final list of food-related businesses was used to filter the review file. For each business, the total number of reviews was then calculated. Only businesses with at least 100 reviews were kept in our final two sample datasets. For Cleveland and Phoenix, there remained a total of $364,336$ reviews from $1,347$ food-related businesses. For Pittsburgh and Toronto, there remained a total of $287,169$ reviews from $1,388$ food-related businesses.

To reproduce this portion of our analysis, please see the instructions contained in the `Yelp_Business_Sample.Rmd` file. Running this file will load in the RDS files obtained from `Yelp_Data_Loadin.Rmd`, will perform the analysis described above, and will then save two sample RDS files: `Cleveland_Phoenix_100.rds` and `Pittsburgh_Toronto_100.rds`. These files were then used in subsequent data wrangling.

1.3 Establishment and Cuisine Type Additions

The `categories` column of our sample datasets contains characteristics or identifiers of a particular business that can be extracted. For our analyses, we extracted information on the cuisine and establishment type.

In order to create a variable for cuisine type, we first created a series of binary variables that indicated if a particular ethnicity was present in the `categories` string for a particular business. Indicators were extracted using functions in the `stringr` package. Then, to create the variable `Type`, the results of these binary variables were used to classify each business into a cuisine. The categories for `Type` are as follows:

1. **European** (British, French, Irish, Spanish, European)
2. **Mexican**
3. **American** (New and Traditional)
4. **Chinese** (Chinese and no other Asian ethnicity)
5. **Italian**
6. **Indian**
7. **Japanese** (Japanese and no other Asian ethnicity)
8. **Mediterranean** (Greek, Mediterranean)
9. **Thai** (Thai and no other Asian ethnicity)
10. **AsianFusion** (AsianFusion, Cantonese, Cambodian, Vietnamese, Korean)
11. **OtherEthnic** (Moroccan, Ethiopian, Hawaiian, Caribbean, Middle Eastern, Peruvian, Persian, Brazilian, Creole, Filipino)
12. **None**

The level `OtherEthnic` contained businesses with an ethnic cuisine indicator (see above), but the total number of businesses with that indicator was less than 10. Businesses were classified as `None` if they contained no ethnic cuisine indicator. The distribution of cuisine type (`Type`) is shown below.

	City	American	AsianFusion	Chinese	European	Indian	Italian
1	Cleveland	79	13	3	12	3	12
2	Phoenix	291	43	32	25	15	107
3	Pittsburgh	141	18	13	23	13	29
4	Toronto	98	73	48	62	21	72

	City	Japanese	Mediterranean	Mexican	None	OtherEthnic	Thai
1	Cleveland	7	6	14	69	6	6
2	Phoenix	36	34	179	309	17	29
3	Pittsburgh	20	14	31	110	10	21
4	Toronto	93	24	35	358	34	27

Table 1.1: Total Number of Cuisine Type by City

Additionally, food-related businesses can fall into multiple establishment types or categories, such as bars or bakeries. Functions in the `stringr` package were used to extract particular words of interest. If a word was present in `categories` for a particular business, then it was classified as that type. The establishment types that were extracted are listed as follows:

1. **Bars**
2. **Bakeries**
3. **Fast Food**
4. **Grocery**
5. **Restaurant**
6. **Other**

The level `Other` contained businesses that did not match any of the other categories. The distribution of establishment type (`food_type`) across the four cities is shown below.

To reproduce this portion of our analysis, please see the instructions contained in the `Yelp_Food_Type.Rmd` file. Running this file will load in the RDS files obtained

	City	Bakery	Bar	Fast Food	Grocery	Other	Restaurant
1	Cleveland	7	104	1	2	13	103
2	Phoenix	45	329	41	15	47	640
3	Pittsburgh	9	162	4	6	25	237
4	Toronto	38	293	19	3	45	547

Table 1.2: Total Number of Each Establishment Type by City

from `Yelp_Business_Sample.Rmd`, will create the `Type` and `food_type` variables, and will then store two sample RDS files: `Cleveland_Phoenix_100_Types.rds` and `Pittsburgh_Toronto_100_Types.rds`. These files were then used in subsequent data wrangling and analysis.

1.4 Business Attributes

The Yelp data set includes business attributes, such as the ambiance, catering, BYOB (Bring Your Own Drink), and more. It also includes the hours of the business. A sample of the raw data can be found below:

```
# object, business attributes to values.
# note: some attribute values might be objects
# "attributes": {
#     "RestaurantsTakeOut": true,
#     "BusinessParking": {
#         "garage": false,
#         "street": true,
#         "validated": false,
#         "lot": false,
#         "valet": false
#     },
# },

# an object of key day to value hours, hours are using a 24hr clock
# "hours": {
#     "Monday": "10:00-21:00",
#     "Tuesday": "10:00-21:00",
#     "Friday": "10:00-21:00",
#     "Wednesday": "10:00-21:00",
```

```
#     "Thursday": "10:00-21:00",
#     "Sunday": "11:00-18:00",
#     "Saturday": "10:00-21:00"
# }
```

However, when reading in the overall business file from the json, these nested dictionaries are then converted to nested dataframes. So to access these variables we need to extract this dataframe, flatten it, and then add it back to the full dataset. A brief overview of both attributes and hours will be given below.

1.4.1 Business Attributes

One difficulty that we faced was that within attributes, there were even more nested dicts. An example can be seen in the original sample, whereby `BusinessParking` has nested variables like `garage`, `street`, etc. On a more technical note, the Yelp json was in a Newline Delimited JSON, which meant that existing R packages like `jsonlite` ran into errors with the newline. Thus, given these difficulties, the extremely large dimension of the dataset already, and the fact that many of these attributes were not available, we decided to remove any further nested attributes. Attributes that were removed were:

```
[1] "BusinessParking"     "GoodForMeal"         "Ambience"
[4] "BestNights"          "Music"               "HairSpecializesIn"
[7] "DietaryRestrictions"
```

All other variables were read in and properly typed as numeric, logical, or factor.

1.4.2 Business Hours

Yelp also provides the hours each business is open for every day of the week. However, it is read in from the json as a nested dataframe, Furthermore, this data is stored in character format, as seen below:

```
     Monday   Tuesday Wednesday  Thursday    Friday  Saturday    Sunday
1 8:30-17:0 11:0-21:0 11:0-21:0 11:0-21:0 11:0-21:0 11:0-21:0      <NA>
2      <NA>      <NA>      <NA>      <NA> 17:0-23:0 17:0-23:0 17:0-23:0
3 10:0-22:0 10:0-22:0 10:0-22:0 10:0-22:0 10:0-22:0 10:0-22:0 10:0-22:0
4      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
```

```
5  8:0-17:0  8:0-17:0  8:0-17:0  8:0-17:0  8:0-17:0      <NA>      <NA>
6 5:30-20:0 5:30-20:0 5:30-20:0 5:30-20:0 5:30-21:0 5:30-21:0 6:30-19:0
```

We believe that the most relevant variable of interest to be extracted is the number of hours per day. To do this, we parsed the the character and turned it into a data type that R understands as time, POSIX. We found the difference between the start and end times, handling edge cases such as when the start and end time were `0:00`, because the business is open for 24 hours.

Technical Notes, if Interested

Another difficulty was related to how dplyr applies such functions, and we found an interesting technical solution. This is of tangential interest to our project and may be skipped. The problem was as follows. dplyr would only apply the function to the first row of the hours, and repeat for the rest, as seen below:

```
  Monday Tuesday Wednesday Thursday Friday Saturday Sunday
1    8.5      10        10       10     10       10      0
2    8.5      10        10       10     10       10      0
3    8.5      10        10       10     10       10      0
4    8.5      10        10       10     10       10      0
5    8.5      10        10       10     10       10      0
6    8.5      10        10       10     10       10      0
```

After some research, we found that this is because dplyr expects the function to be vectorized, or able to apply to all rows in one function call. One solution was to apply the function to each row, such as by using `rowwise()`. However, Hadley Wickham himself argued that this is an inefficient method, and instead argued that one should vectorize functions being applied to dataframes.

Thus, after vectorizing the function, we were able to extract all hours from the dataset, as seen below:

```
  Monday Tuesday Wednesday Thursday Friday Saturday Sunday
1    8.5    10.0      10.0     10.0   10.0     10.0    0.0
2    0.0     0.0       0.0      0.0    6.0      6.0    6.0
3   12.0    12.0      12.0     12.0   12.0     12.0   12.0
4    0.0     0.0       0.0      0.0    0.0      0.0    0.0
5    9.0     9.0       9.0      9.0    9.0      0.0    0.0
6   14.5    14.5      14.5     14.5   15.5     15.5   12.5
```

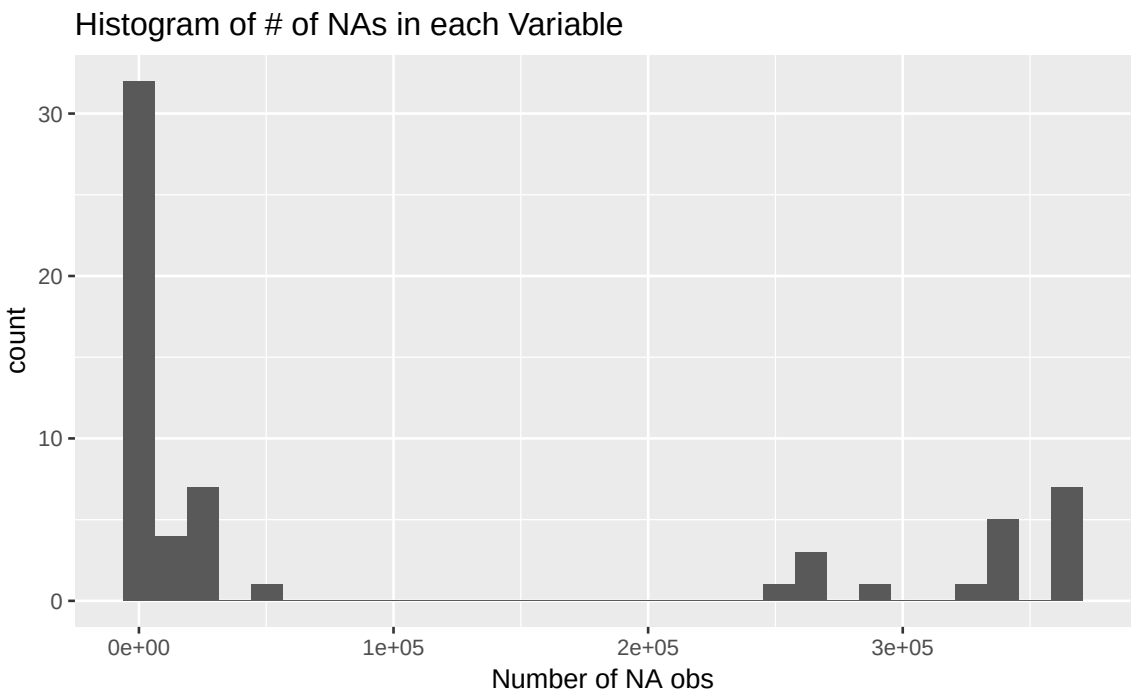


Figure 1.1: Number of NA obs in each Variable

1.5 Data Cleaning

When processing this dataset, we noticed that some variables had a high number of NA observations, as seen below:

There is a clear break whereby some variables have greater than $2 * 10^5$ NA observations whereas most variables have NA observations less than $1 * 10^5$. Given that our dataset itself only contains roughly $7 * 10^5$ observations, we decided to remove these variables from consideration during modelling. The variables removed are:

RestaurantsAttire	Alcohol
37	43
DogsAllowed	DriveThru
45	46
WiFi	BYOB
49	50
BYOBCorkage	CoatCheck
51	52
Corkage	GoodForDancing
53	54
HappyHour	Smoking

```
                        55                         56
       ByAppointmentOnly          AcceptsInsurance
                        57                         58
   BusinessAcceptsBitcoin               AgesAllowed
                        59                         60
RestaurantsCounterService               Open24Hours
                        61                         62
```

The numbers below each variable are the index for the variable in the dataframe and can be ignored. From this, we can see that many business attribute variables have missing information, and also goes to show the level of detail Yelp collects on businesses - even whether the Business accepts Bitcoin.

1.6 The Final Dataset:

For our analyses, the original Yelp competition data was reduced by the above criteria to produce two smaller samples containing important food-related business and review information. The final dataset from Cleveland and Phoenix (`Cleveland_Phoenix_Full.rds`) contains a total of $364,336$ reviews and $1,347$ businesses, each with 100 or more reviews. The dataset from Pittsburgh and Toronto (`Pittsburgh_Toronto_Full.rds`) contains a total of $287,169$ reviews and $1,388$ businesses, also each with 100 or more reviews. Each dataset includes the following variables:

1. **business_id**: the unique identifier for a particular business
2. **review_id**: the unique identifier for a particular review
3. **user_id**: the unique identifier for the user who wrote the review
4. **stars_review**: the number of stars given in a review $(0, 1, 2, 3, 4$ or $5)$
5. **stars_business**: the average star value for a business $(0, 0.5, 1, ..., 4.5$ or $5)$
6. **date**: the date of the review
7. **text**: the actual text, or written portion, of the review
8. **useful**: the number of users that found the review useful
9. **funny**: the number of users that found the review funny

10. **cool**: the number of users that found the review cool
11. **name**: the name of the business
12. **neighborhood**: the neighborhood where the business is located
13. **address**: the address of the business
14. **state**: the state where the business is located
15. **postal_code**: the postal code of where the business is located
16. **latitude**: of the business
17. **longitude**: of the business
18. **is_open**: whether or not the business is open (1) or permanently closed (0)
19. **categories**: descriptors of the business, such as "Breakfast" or "Shopping"
20. **Type**: identification of cuisine type including Italian, American, Chinese, and more
21. **review_count**: the number of reviews a particular business received
22. **city**: the city where the business is located
23. **food_type**: identification of business type, including Bar, Bakery, Fast Food, Grocery, Restaurant, and Other (factor)
24. **Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday**: Respectively, the number of hours it was open for each of these dates
25. **BikeParking**: whether the business allows patrons to park bikes
26. **BusinessAcceptsCreditCards**: whether the business accepts credit cards
27. **GoodForKids**: whether the business is good for kids
28. **HasTV**: whether the business has a TV
29. **NoiseLevel**: factor with four levels: `average, loud, quiet, very_loud`
30. **OutdoorSeating**: whether the business has outdoor seating
31. **RestaurantsDelivery**: whether the restaurant delivers

32. **RestaurantsGoodForGroups**: whether the restaurant is good for groups

33. **RestaurantsPriceRange2**: a numeric variable, integers from 1 to 4, representing the number of $ signs. The number of dollar signs correspond with the per person price range of the restaurant: $ = $1-10, $$ = $11-30, $$$ = $31-60, $$$$ = $61+

34. **RestaurantsReservations**: whether the restaurant takes reservations

35: **RestaurantsTakeOut**: whether the restaurant has takeout

36: **Caters**: whether the restaurant caters

37: **RestaurantsTableService**: whether the restaurant has table service

38: **WheelchairAccessible**: whether the business is wheelchair accessible

In our analyses, sentiment analysis will be performed on the `text` variable and the extracted sentiments will subsequently be used to predict `stars_review`. Thus, our response variable of interest is the stars a business received from a particular review. The distribution of star rating from all reviews in our sample, by city, are shown below.

	City	1	2	3	4	5
1	Cleveland	3942	4207	6386	14053	20707
2	Phoenix	27553	25455	35421	81491	145121
3	Pittsburgh	6916	8192	13797	29493	37586
4	Toronto	13629	18159	34885	66514	57998

Table 1.3: Total Number of Reviews with a Particular Star Rating by City

For each city, the number of reviews increases as the star category increases. For example, in Phoenix, of a total of $315,041$ reviews, only 8.75% of them were had a star rating of 1 compared to 46% of reviews with a star rating of 5. We would anticipate, then, that positive sentiment values will be more common in subsequent analysis.

Chapter 2

Sentiment Analysis

Our goal was to utilize sentiment analysis and work with several different lexicons to determine which performed best at associating negative/positive words with certain reviews. The big idea behind a lexicon is to move from considering text as a combination of individual words to considering the sentiment of text as a combination of sentiments of individual words.

2.1 tidytext and Sentiment Lexicons

The **tidytext** package contains several sentiment lexicons in the sentiments dataset. These lexicons are general-purpose as they are based on single words and assign scores to these words for positive or negative sentiment. The lexicons included are:

1. **AFINN** - assigns scores to each word ranging from -5 (strongest negative sentiment) to 5 (strongest positive sentiment).
2. **bing** - categorizes words in a binary fashion into positive and negative categories.
3. **nrc** - categorizes words in a binary fashion into several categories (positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust).
4. **loughran** - categorizes words in a binary fashion into several categories (positive, negative, litigious, uncertain, constraining, and superfluous).

An important aspect of these lexicons is that they do not take qualifiers into account (i.e. "the food was **not** good"). This is because these lexicons are based on single words.

The AFINN, bing, and nrc lexicons are validated on restaurant and movie review data, making them very suitable for this analysis. Loughran on the other hand is

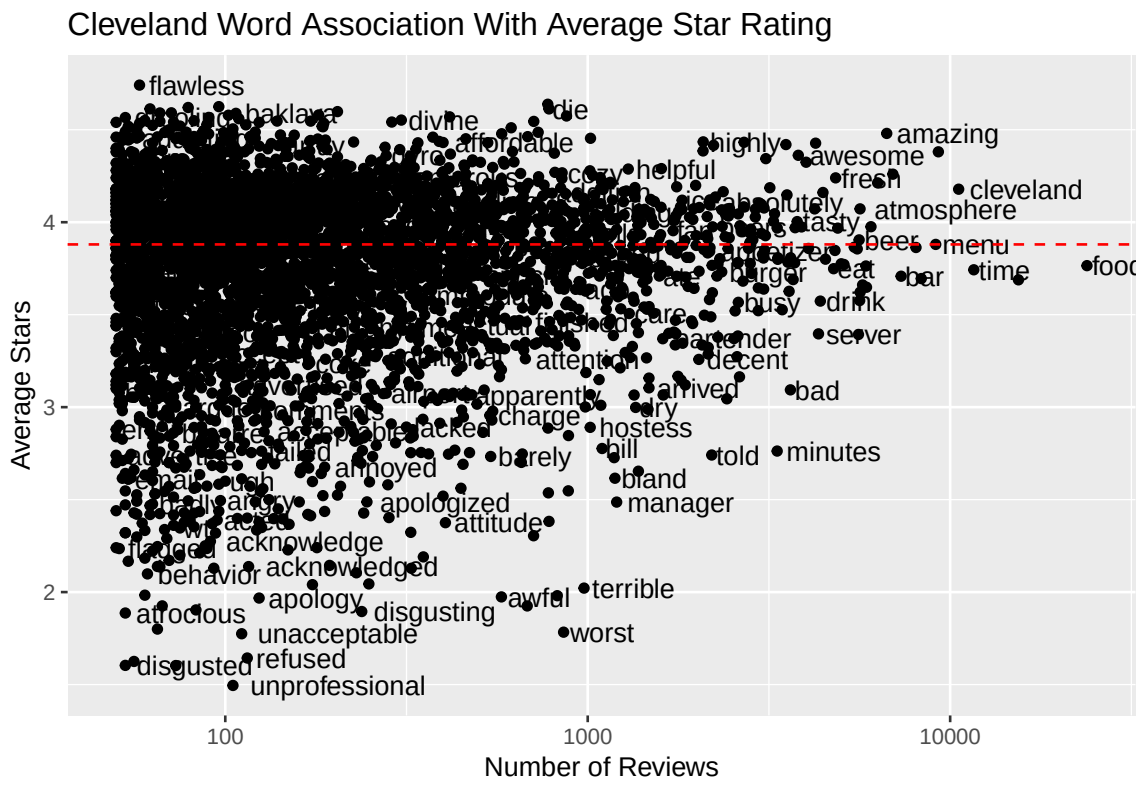


Figure 2.1: Cleveland Word Association

validated on financial data, so we were interested to see how it would perform on this dataset.

2.2 Graphs and Lexicon Comparison

2.2.1 Cleveland vs Phoenix

Our first comparison is between the cities of Cleveland, Ohio and Phoenix, Arizona. We chose these cities because they are located on different sides of the United States and they are fairly different culturally. We chose to utilize solely the AFINN lexicon in the exploratory plots because it is popular and has a different scoring-style than the other lexicons.

Word Association

Figures 2.1 and 2.2 demonstrates how certain words contained in reviews are associated with the 5-star rating. For example, in the Cleveland plot, we can see that words like "unprofessional" and "worst" correspond to a star-rating below 2. We also see that

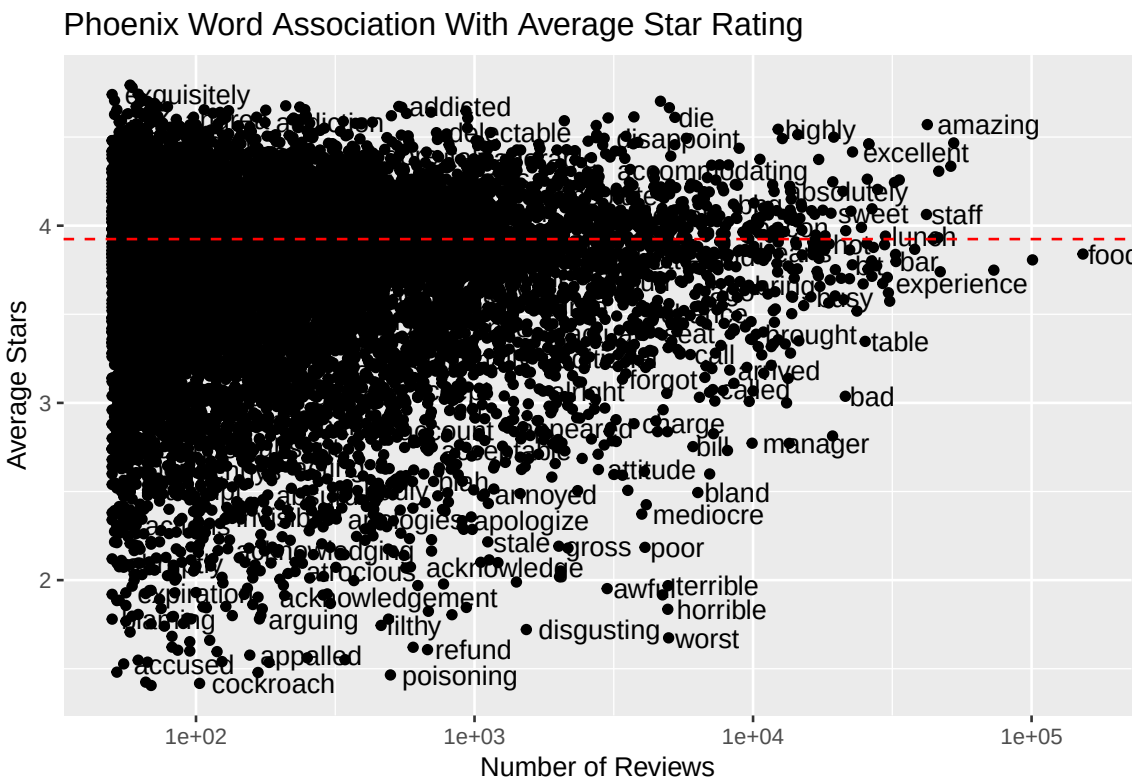


Figure 2.2: Phoenix Word Association

words like "amazing" and "delicious" appear in reviews often and correspond to a 4.5-star rating on average. It is also important to note that Cleveland restaurants have an average star-rating of about 3.85.

Phoenix restaurants on the other hand have an average star-rating of about 3.95. Words like "cockroach" and "poisoning" suffer star-ratings below 2 while words like "addicted" and "amazing" rack up between 4 and 4.5 stars.

AFINN Overpredictions and Underpredictions (Part 1)

Figures 2.3 and 2.4 demonstrate how the AFINN lexicon is over- and under-predicting certain words. Towards the left of the plots, around an AFINN score of -4, we can see a group of profanities are associated with low-star rated reviews.

However, the word "damn" is actually associated with higher-star ratings than the rest of those words. Because it is a type of curse word, AFINN gives "damn" the same score as the rest of the curse words. However, "damn" is often used in a positive context to convey the extent of certain characteristics (i.e. "the chocolate cake was so **damn** good I almost cried!"). This is an instance where the lexicon fails, as it cannot put the word into context and thus under-predicts its value.

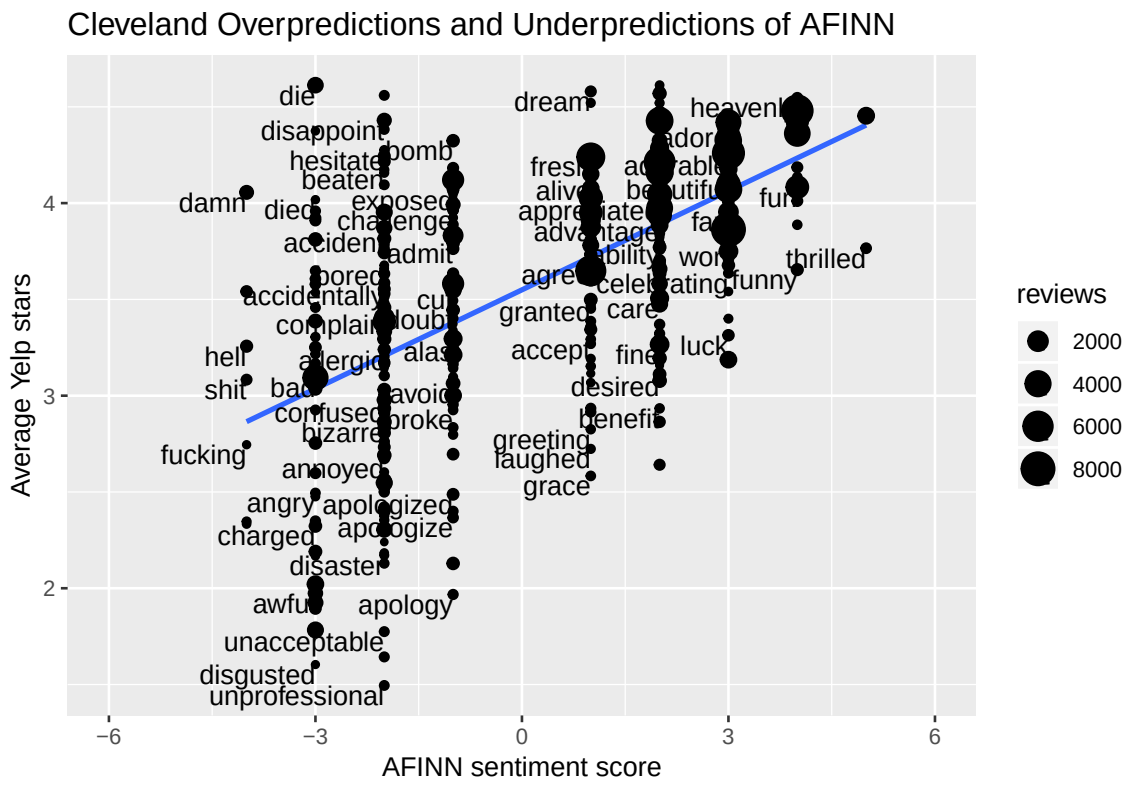


Figure 2.3: Cleveland AFINN Overpredictions and Underpredictions

AFINN Overpredictions and Underpredictions (Part 2)

Figures 2.5 and 2.6 are another way to compare the lexicon's sentiment predictions to the actual star-rating. In Cleveland's plot, words like "die" and "disappoints" are scored low by AFINN but correspond to a high star-rating ("the chicken Parmesan is to **die** for! It never **disappoints**!"). Words like "joke" are scored more positively by AFINN but correspond to lower star-ratings ("the service was a **joke**").

In the Phoenix plot, words like "bomb" and "forget" are scored low by AFINN and correspond to high star-ratings ("this place is the **bomb**! I'll never **forget** how delicious the food was"). Words like "luck" are scored more positively by AFINN but correspond to lower star-ratings ("good **luck** with this place - it took the servers an hour to bring out our food").

These plots are a great visual representation to use when trying to understand how the lexicon is performing with respect to star rating. If the lexicon was working perfectly, we would expect to see a gradient from blue/purple to red/orange starting from the top of the plot at 5 stars. Instead, we see some sort of a mix of colors, although the lexicon does perform relatively well.

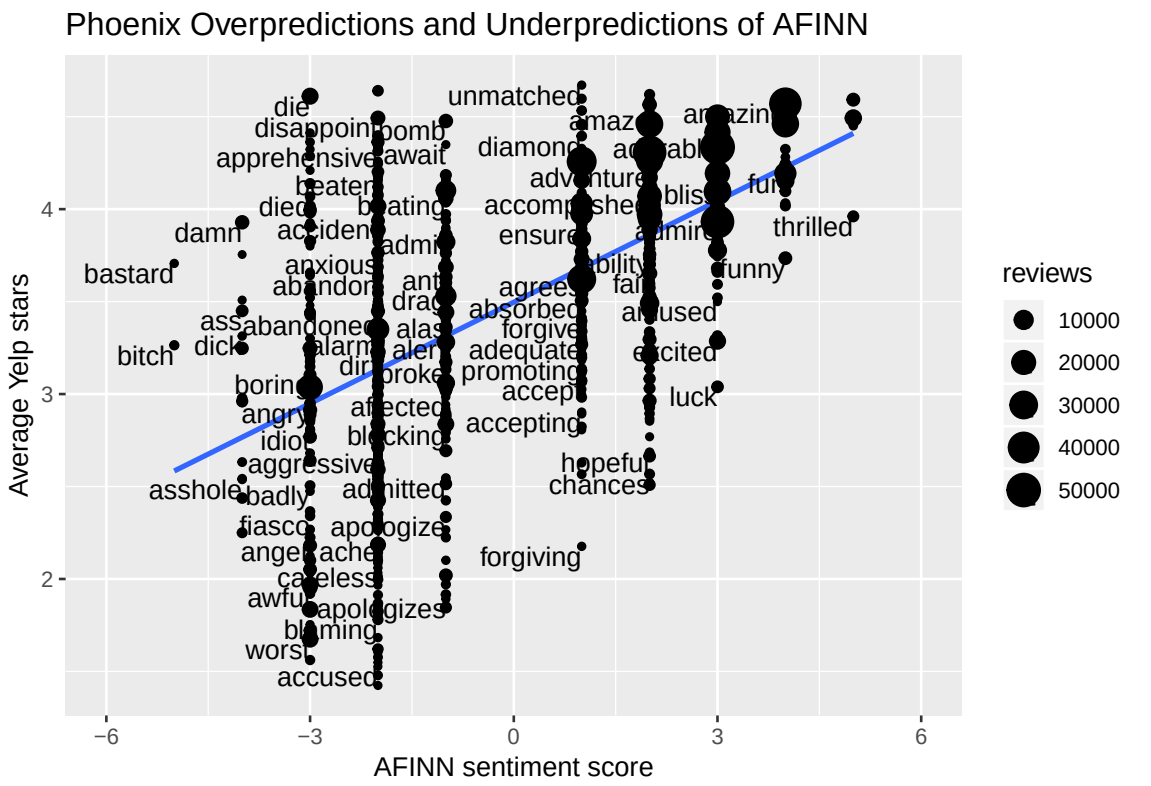


Figure 2.4: Phoenix AFINN Overpredictions and Underpredictions

Lexicon Comparisons

Figures 2.7 and 2.8 display the performance of each of the four lexicons on both datasets. Although there are outliers, in both cities it seems that the lexicons performed similarly in associating the sentiment of reviews with the star-rating of a restaurant. Although the loughran lexicon stands out and seems to have performed the worst. This is not surprising because this lexicon was validated on financial data, while the other three lexicons were validated on restaurant and movie review data.

2.2.2 Toronto vs Pittsburgh

As a small analysis on the side, we were also interested in comparing a Canadian city with a city in the United States. We chose to compare Toronto, Ontario and Pittsburgh, Pennsylvania.

This complementary analysis was fueled by a somewhat common idea that "Canadians are nicer" than the average person. On a very basic level, we examined the average star-rating for all restaurants in both cities.

Figure 2.9 shows that Pittsburgh's restaurants have an average star rating of 3.86

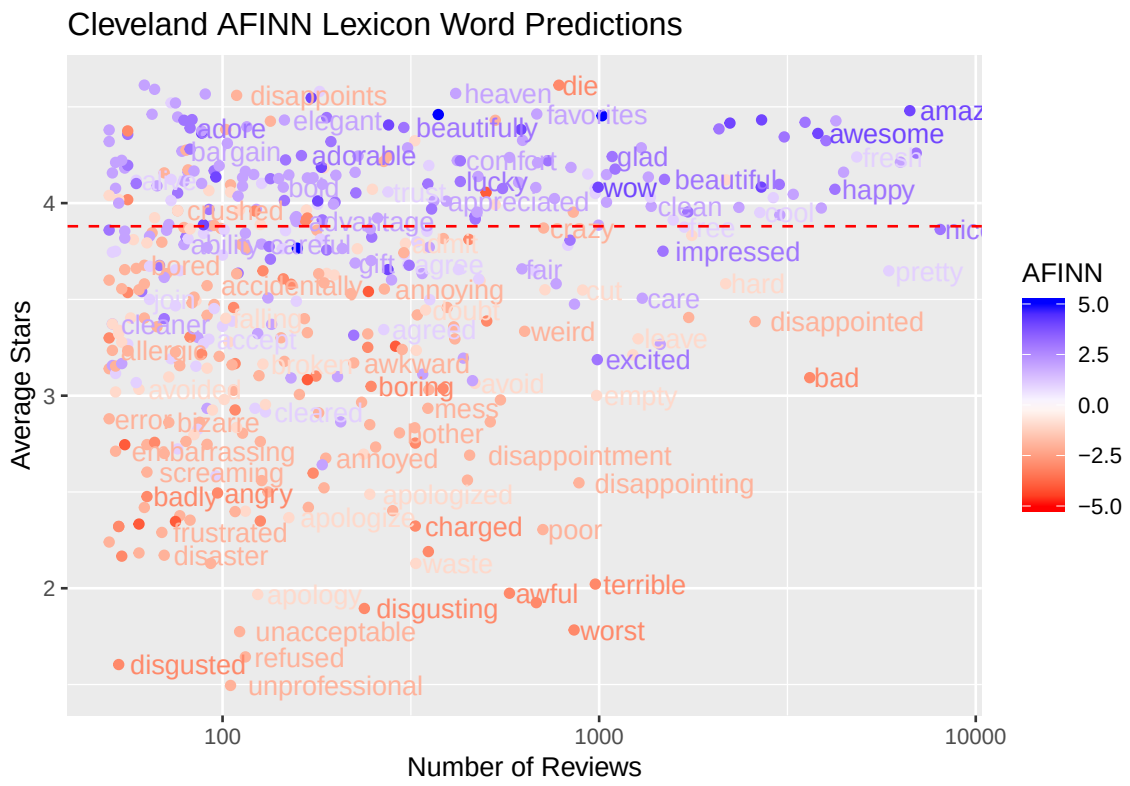


Figure 2.5: Cleveland Colored Word Plots

while Toronto's restaurants have an average star rating of 3.72.

In addition, we also explored the average star rating by cuisine type in each city. The cuisine types included are:

1. **American**
2. **AsianFusion**
3. **Chinese**
4. **European**
5. **Indian**
6. **Italian**
7. **Japanese**
8. **Mediterranean**
9. **Mexican**
10. **OtherEthnic**
11. **Thai**

Figure 2.10 below shows that, on average, Pittsburgh restaurants of various food types have a higher star rating than those in Toronto. The only category in which

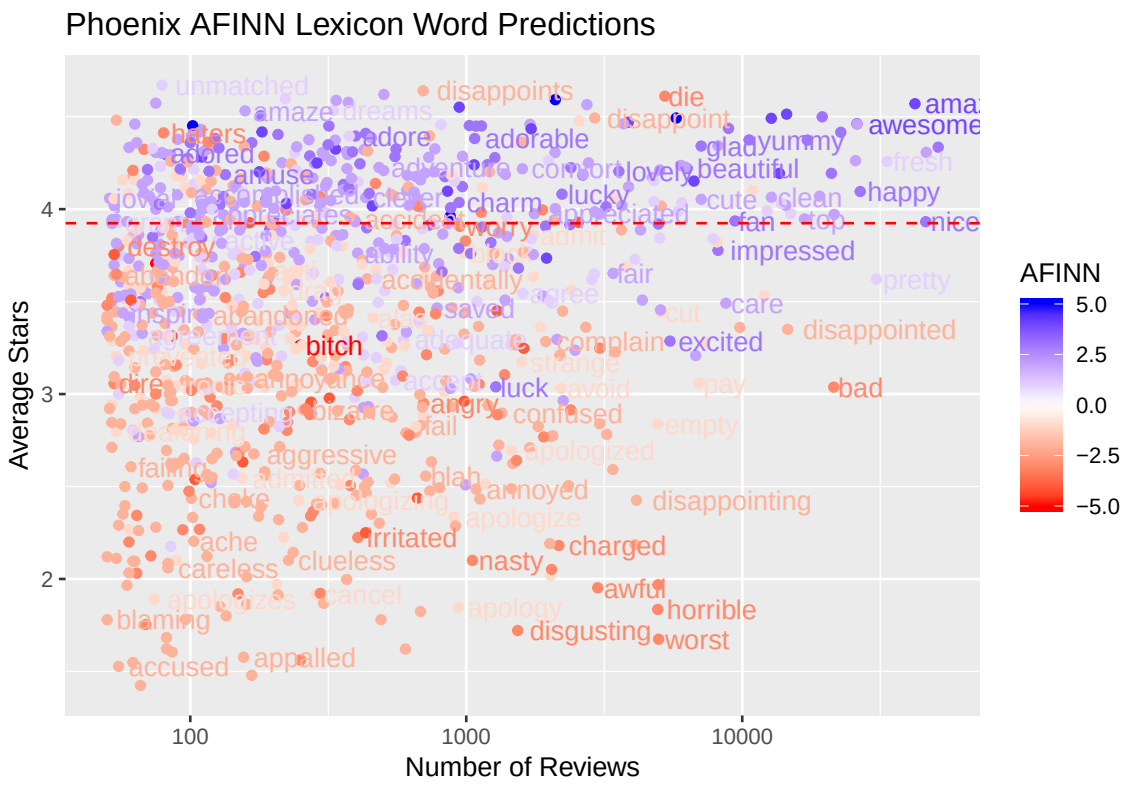


Figure 2.6: Phoenix Colored Word Plots

Toronto receives a higher average star rating than Pittsburgh is Japanese cuisine.

We can also examine the most positive and negative words associated with the reviews in each city.

In these outputs from Toronto, we can see that the top positive word is “unmatched” (4.65 stars) while the top negative word is “rudest” (1.30 stars).

The word summaries for Pittsburgh show that the top positive word is “Argentina” (4.68 stars) while the top negative word is “disrespectful” (1.56 stars).

The word summaries from Toronto do not reveal any sort of inclination for reviewers to be “nice”; from this preliminary analysis it seems Canadian reviewers are just as opinionated and honest as American reviewers.

This was a complementary analysis that we would have explored further if time permitted. As outlined in our proposal, we simply wanted to investigate whether the idea that “Canadians are nicer” would be upheld or represented in the Yelp reviews. This preliminary analysis is inconclusive, and further investigation may involve larger datasets (comparison of several Canadian and U.S. cities) and new approaches to contextual sentiment analysis.

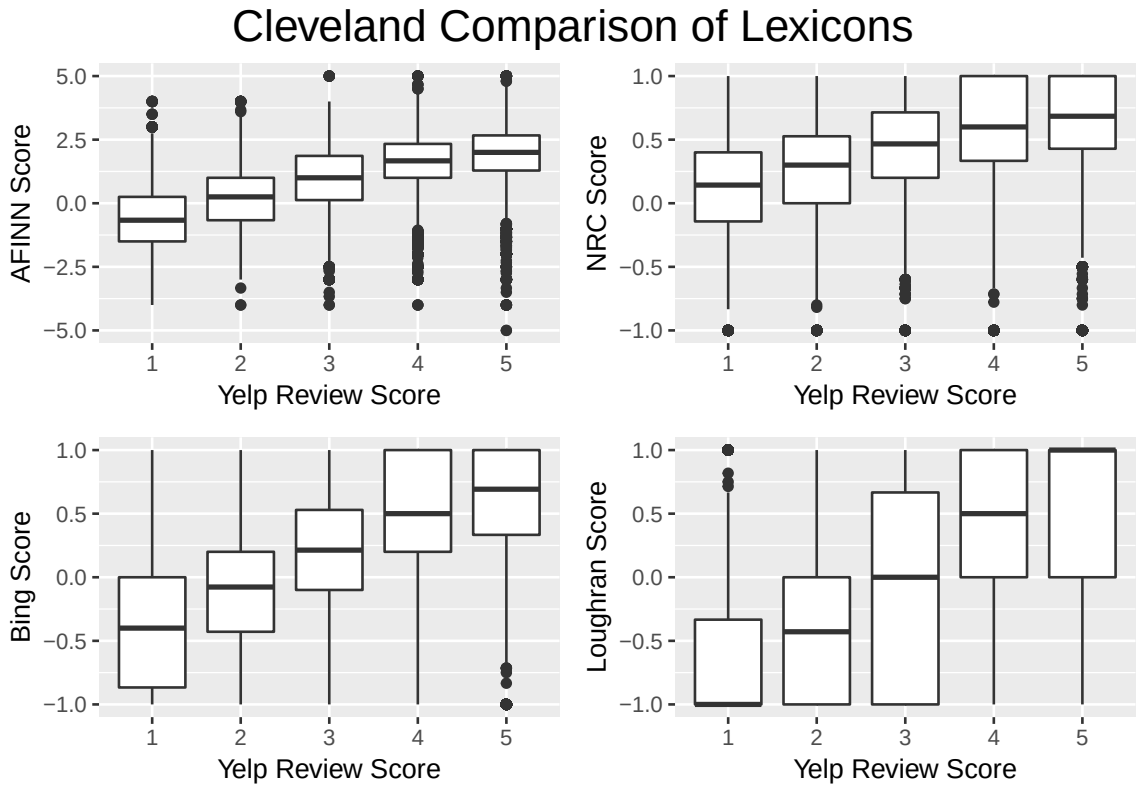


Figure 2.7: Comparing Lexicon Performance on Cleveland Dataset

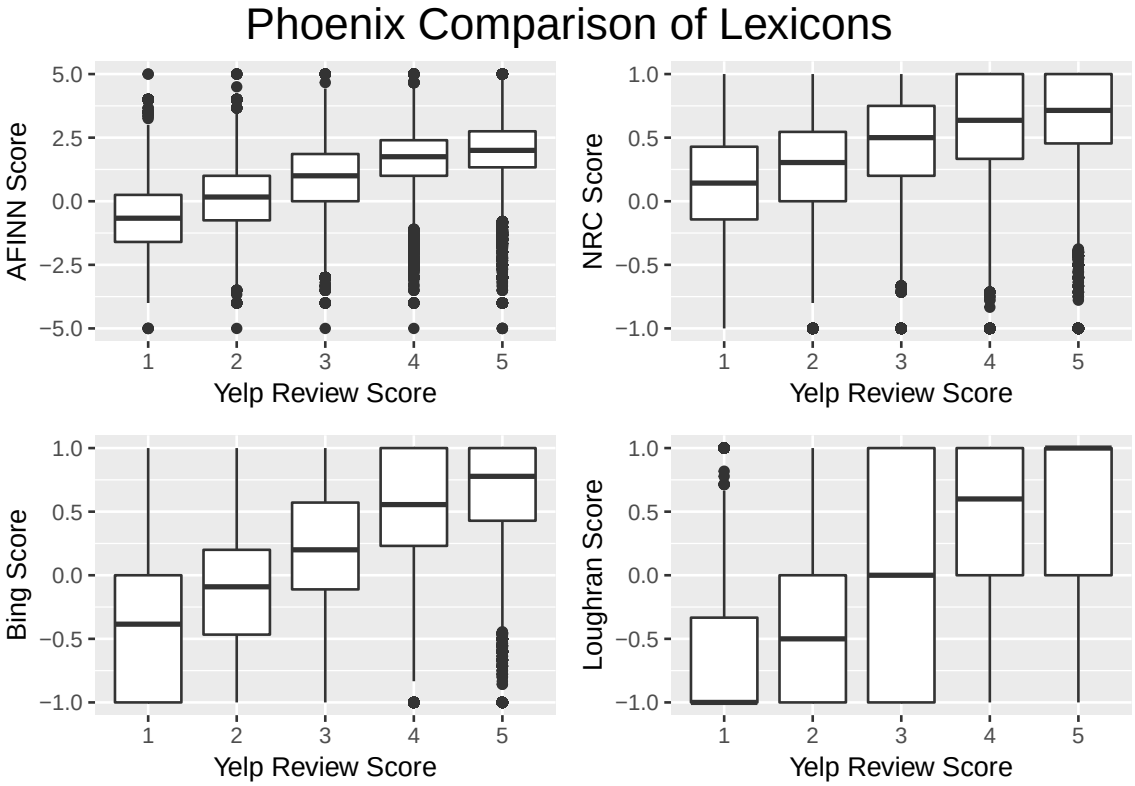


Figure 2.8: Comparing Lexicon Performance on Phoenix Dataset

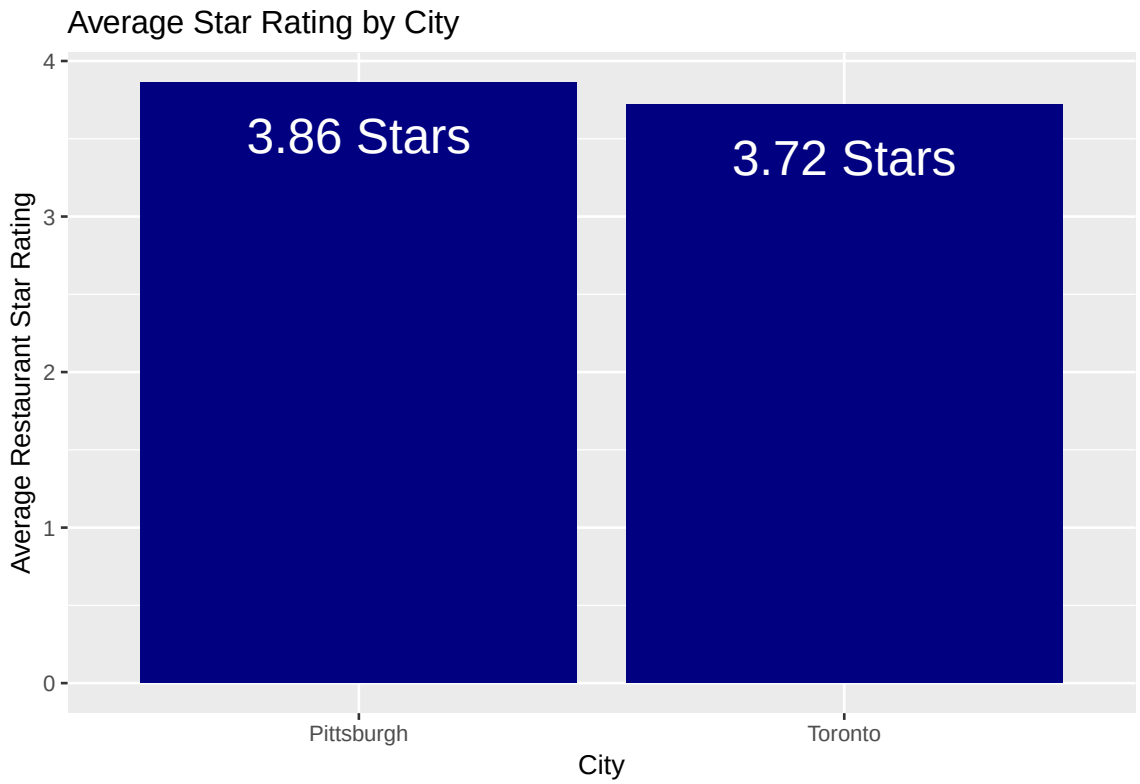


Figure 2.9: Comparing Average Star Rating in Pittsburgh and Toronto

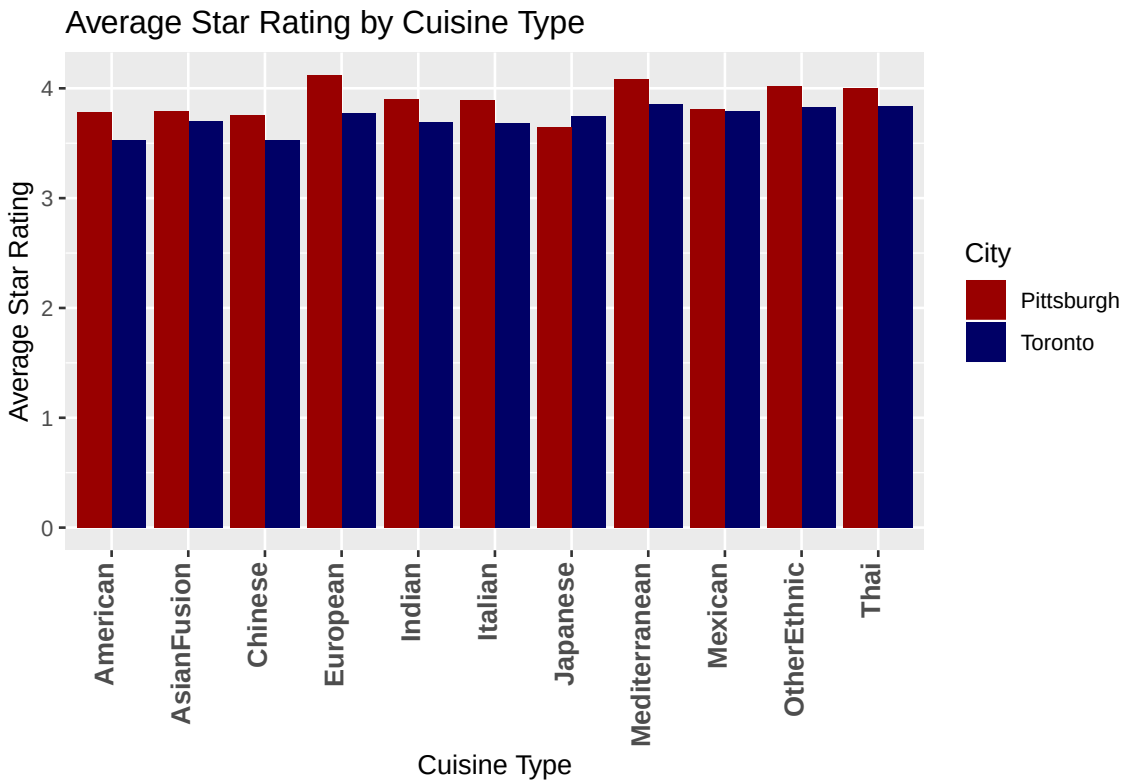


Figure 2.10: Comparing Average Star Rating by Restaurant Type in Pittsburgh and Toronto

Chapter 3

Models

3.1 Overview

Our response variable, `stars_review`, has 5 levels from 1-5. Thus, to predicting `stars_review` is a multi-class classification problem. This unfortunately means that binary classification methods we learned in class like logistic regression, boosting, etc. do not apply in our situation. There are multi-class versions of these algorithms. However, we will be applying algorithms that naturally extend to multi-class classification problems, Classification Trees and Random Forests. We believe that Classification Trees will allow us to most directly meet our objectives, which are to determine which of the four lexicons was most accurate, and to see if there are any significant differences between the East and West of the US. Finally, the Random Forest is a natural extension, can be potentially more accurate, and can help us parse out the marginal impact of each variable.

First, we fit a Classification Tree on each lexicon, in order to see individual differences between their predictive power and categorization. If any lexicon stands out, or any lexicon performs particularly poorly, we might reflect those differences when constructing further models.

Second, we fit a Classification Tree on all lexicons. By including all sentiment predictors in the same model, we can see if the predictors themselves are highly correlated. We would expect that bing would definitely be included in this tree, but with all sentiment scores we can see if any other sentiment scores have extra information to add.

Third, we fit a Classification Tree on all predictors, and tune the tree to make it bigger. These model is to see if there are any interactions between business attributes and sentiment, and to see whether sentiment or attributes are more powerful predictors.

However, changes were made to the training set, due to a variety of painful technical issues.

Finally, we fit a Random Forest on all predictors. This model is to see if we can benefit from its properties of variance reduction through averaging many deep trees and achieve a higher accuracy. We also show the marginal impact of individual predictors.

3.1.1 Baseline Accuracy.

The underlying response variable we are trying to predict here `stars_review`, is highly skewed, as can be seen below:

```
         1          2          3          4          5
0.08644717 0.08141306 0.11474960 0.26223735 0.45515282
```

One simple baseline that can be used to assess our future models is majority prediction - if we predict all reviews to be 5, then we would be right 45.5% of the time.

Furthermore, to assess the accuracy of the models I created a function `AssessAccuracy` and `AssessAccuracyRF`, which outputs by default only the percentage of correct predictions on the test set, for respectively Classification Trees and Random Forests. 'AssessAccuracyRF

3.2 Dataset Preparation

From the complete dataset, certain variables that are specific to each business, review, or user were dropped. The variables dropped are: 'business_id', 'review_id', 'user_id', 'name', 'state', 'date', 'address', 'text', 'categories'

State was dropped because we have already split the data into just two cities. Date was dropped because it didn't seem particular relevant to our analysis. Text was also dropped because we will predicting based on the sentiment in the text, not the text itself. Categories is dropped because we have already parsed out the categories into `food_type` and `Type`.

The data was split into a training set and test set, with the training set containing 90% of the full data set. The training set contains 327883 observations and 39 predictors. The test set contains 36432 observations and 39 predictors.

3.3 Classification Trees

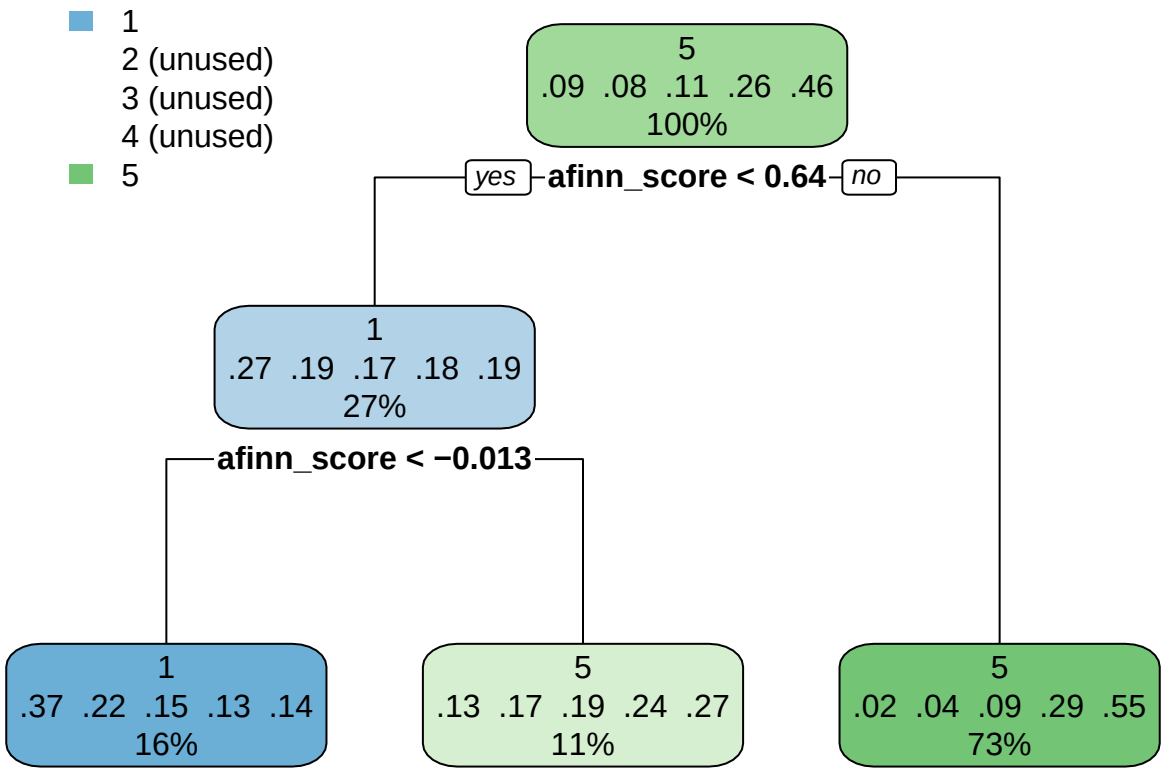
3.3.1 Fitting on Individual Sentiment Lexicons

We first fit a tree on each individual sentiment lexicon to see individual differences between their predictive power and categorization. If any lexicon stands out, or any lexicon performs particularly poorly, we might reflect those differences when constructing further models.

AFINN

Below is an example for how we fit a `rpart` classification tree on the a sentiment predictor. Further models will simply print the output, if the reader wants to examine the code, it is in the Rmd.

```
afinn.rpart <- rpart(stars_review ~ afinn_score, data = train, method = "class")
rpart.plot(afinn.rpart)
```



From this plot of the classification tree, we can see that the tree completely ignores reviews of 2,3, or 4, only predicting either 1 or 5. This reflects in part the underlying skewed nature of the review distribution, but also goes to show how AFINN is good at predicting extremes, either a terrible review, which would be 1, or a stellar review, which would be 5. This is also demonstrated with the end nodes. For example, the middle end node, which contains reviews with $-0.013 < afinn\_score < 0.64$ has a distribution of star ratings from 1 to 5, respectively, of 0.13, 0.17, 0.19, 0.24 and 0.27. This is a relatively even distribution that increases only slightly, which goes to show that the AFINN score and review count is not highly correlated for scores that are close to neutral.

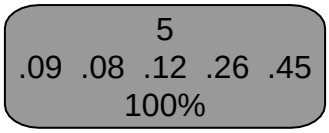
We now assess the accuracy of the classification tree with the `AssessAccuracy` fiction. Future assessments will just show output.

```
AssessAccuracy(afinn.rpart)
```

```
[1] "Percentage Correct:  0.487099"
[1] "Percentage Wrong:  0.512901"
```

This model predicted the right review roughly 48.7% of the time, which is only a minor 3% improvement over the baseline of 45.5%.

nrc



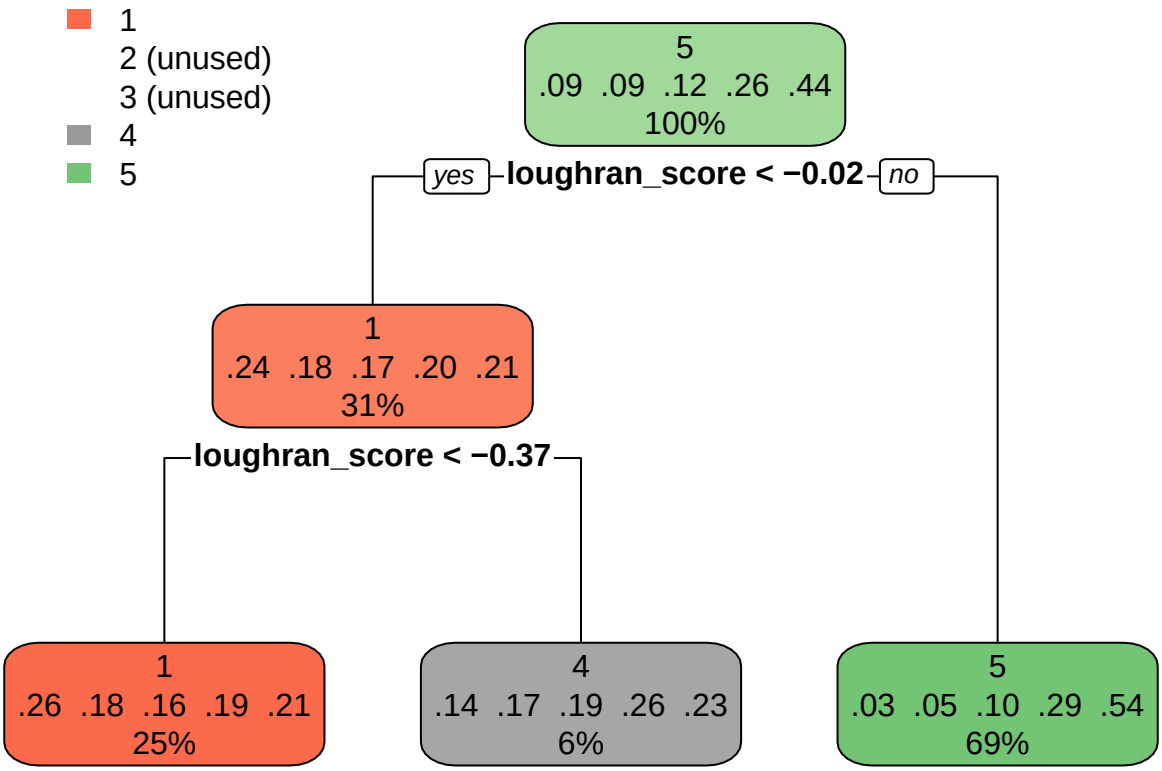
This model completely failed to use the nrc score, and predicted all reviews to be 5. This is a clear indication that there is a poor correlation between the nrc score and review.

We now assess accuracy.

```
[1] "Percentage Correct:  0.456577"
[1] "Percentage Wrong:  0.543423"
```

Since this model ignored the nrc score, the accuracy is simply the baseline, or 45.7%. This baseline is slightly different from the full dataset, as it was assessed on the test data set.

loughran



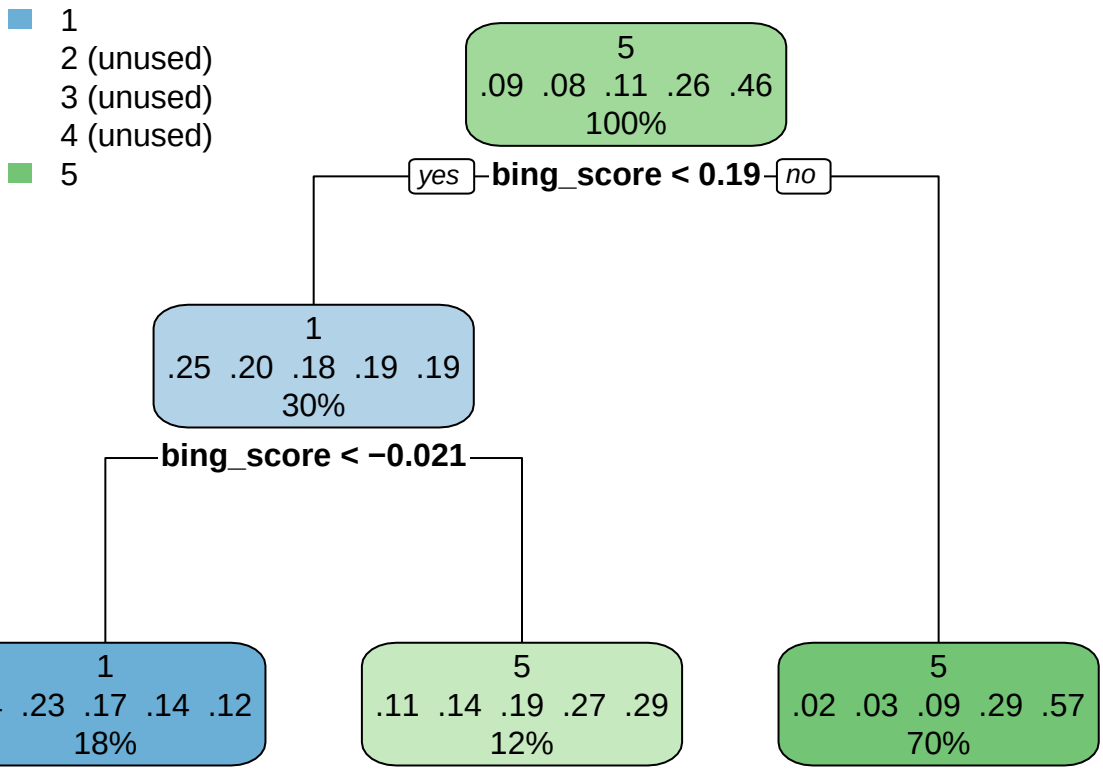
From this plot of the classification tree, we can see that the tree completely ignores reviews of 2 and 3, only predicting 1, 4 or 5. Thus, the results are somewhat similar to the AFINN model. This is also demonstrated with the end nodes. For example, the middle end node, which contains reviews with $-0.37 < loughran\_score < -0.02$ has a distribution of star ratings from 1 to 5, respectively, of 0.14, 0.17, 0.19, 0.26 and 0.23. Again, similarly to AFINN, this is a relatively even distribution that increases only slightly, which goes to show that the loughran score and review count is not highly correlated for scores that are close to neutral.

We now assess accuracy.

```
[1] "Percentage Correct:  0.465882"
[1] "Percentage Wrong:  0.534118"
```

What is uprising here is that even with a more detailed separation than AFINN, this model performs roughly 2% worse, with an accuracy rate of 46.5% compared to AFINN's 48.7%. This suggests that loughran may be a poor predictor compared to AFINN.

bing



From this plot of the classification tree, we can see that the tree completely ignores reviews of 2,3, or 4, only predicting either 1 or 5. Thus, the results are somewhat similar to the AFINN model. However, closer examination of the end nodes shows a slight difference. For example, the middle end node, which contains reviews with $-0.021 < bing\_score < 0.19$ has a distribution of star ratings from 1 to 5, respectively, of 0.11, 0.14, 0.19, 0.27 and 0.29. Compared to the AFINN middle node, with a distribution of star ratings from 1 to 5, respectively, of 0.13, 0.17, 0.19, 0.24 and 0.27, we can see that bing's classification is able to differentiate reviews slightly more than AFINN.

We now assess accuracy.

```
[1] "Percentage Correct:  0.492726"
[1] "Percentage Wrong:  0.507274"
```

This minor improvement in differentiation is reflected in the accuracy of 49.3%, higher than all previous lexicons, and a roughly 1% improvement over the AFINN accuracy of 48.7%

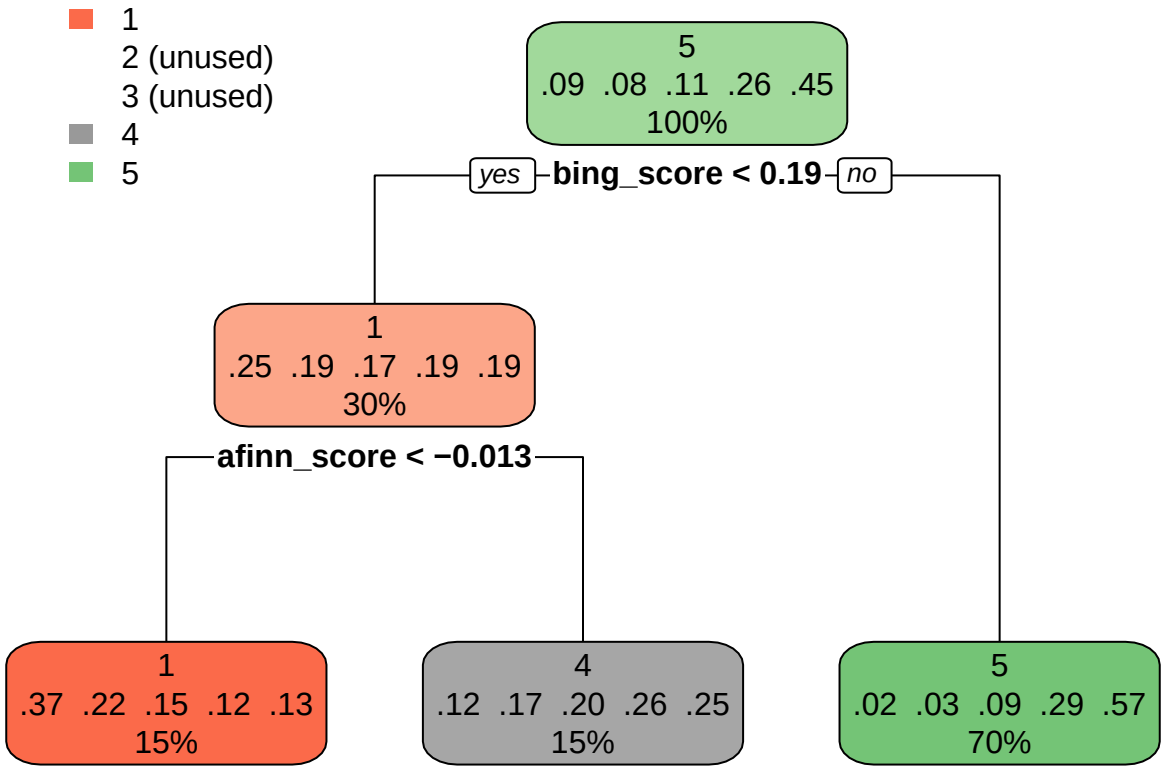
Conclusion

From fitting classification trees individually on each lexicon, we can see that sentiment scores only marginally improve on the baseline prediction of 45.5%. The trees largely ignored the intermediate review scores, 2,3 and 4, choosing to separate only the extremes of 1 or 5. Even still, we can see from the end nodes that there are still significant errors. For example, in every tree the right most node, which predicts the review is 5 stars, had roughly 30% reviews that were 4 stars. On the left most node, which predicts that the review is 1 stars, we can see that the percentage of reviews that were in fact 5 stars ranged from 12% to 21%.

Thus, from this first pass we can conclude that it is extremely difficult to parse out any intermediate scores, 2,3, or 4, and that even by only predicting the extremes, there are significant errors. This reflects the underlying noisiness and skewed distribution of the sentiment scores, and a poor correlation with review stars.

3.3.2 All Sentiment Predictors

By including all sentiment predictors in the same model, we can see if the predictors themselves are highly correlated. We would expect that bing would definitely be included in this tree, but with all sentiment scores we can see if any other sentiment scores have extra information to add.



From the plot, we can see that only bing and AFINN sentiment scores were used, which makes sense given that they were both the highest scoring lexicons when fitting individual trees. Furthermore, this shows that AFINN does have extra information compared to the bing score, which is promising. However, many similarities still remain. This includes the tree ignoring 2 and 3, choosing to only predict either 1, 4, or 5. The middle end node as well has a relatively even distribution of star ratings from 1 to 5, respectively, of 0.12, 0.17, 0.20, 0.26 and 0.25.

We now assess accuracy.

```
[1] "Percentage Correct:  0.491299"
[1] "Percentage Wrong:  0.508701"
```

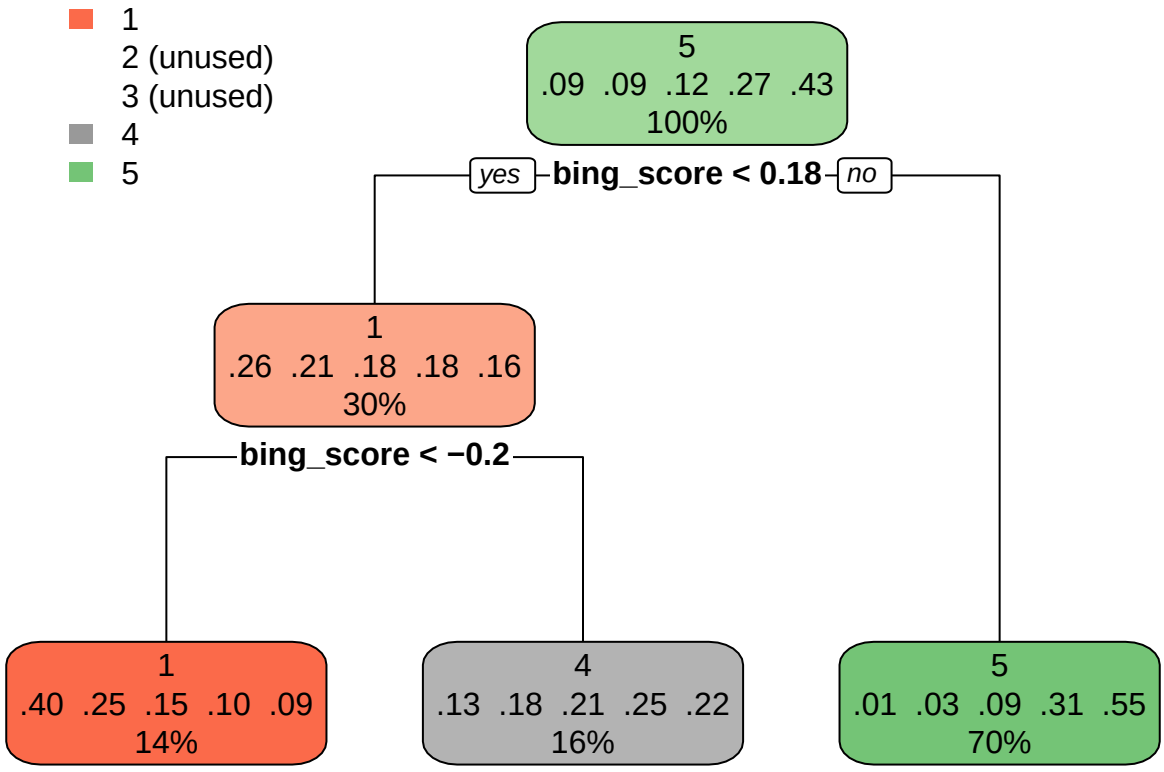
Surprisingly, even with two different sentiment scores as predictors, the accuracy of this model, at 49.1% is slightly lower than the bing model, which had an accuracy of 49.3%. This may be due to minor differences in the training set and test set, as theoretically if the only bing tree was more accurate than the combined bing AFINN tree then it should output the only bing tree. Overall, this result is somewhat disappointing, as we again achieve only a marginal improvement over the baseline majority prediction of 45.5%.

3.3.3 Kitchen Sink Tree

We now fit a kitchen sink tree, with all of the predictors in the dataset. This model is to see if there are any interactions between business attributes and sentiment, and to see whether sentiment or attributes are more powerful predictors. However, changes were made to the training set, due to a variety of painful technical issues.

We were confronted with `rpart` continuously causing the R session to crash, requiring tedious R session restarts. First, we tried removing all NA observations. rpart has a `na.action` parameter, whereby "the default action deletes all observations for which y is missing, but keeps those in which one or more predictors are missing", but we theorized that this would slow the model building down. However, rpart continued to crash, and so we decided to reduce the number of observations to speed up the process. When even a reduction to 0.01 of the original training set, which had only 2000 observations, continued to cause rpart to crash, we decided to edit the dataset.

After some research, we found that the most likely reason was factor variable had too many levels. The worst offender in this case was `postal_code`, which had 66 levels. Thus, to reduce the number of levels, we found the top 20 postal codes, and transformed the rest of the postal codes to one level, `-1000`, to make sure it didn't overlap with any existing factors. We renamed this variable to `postal_shrunk`, and dropped the original `postal_code` predictor from the dataset. R, however, continued to crash, and so we also removed all NA observations. This reduced dataset we renamed to `train.small`



Surprisingly, we have essentially returned to the bing only tree, with only minor differences in the cutoff parameters. This means that the other 37 predictors did not contribute significantly to the review score. We can take a closer look at selection process by looking at variable importance, shown here:

```
#  bing_score    afinn_score loughran_score      nrc_score
# 12056.71272     6294.61044     4403.94763     3845.12462

#  nrc_score stars_business   review_count   neighborhood
#  220.08079       18.01398       10.67754
```

From this, we can tell that almost all aspects of the restaurant itself are insignificant compared to the sentiment predictors. Bing was the clear winner by far, and AFINN, loughran, and nrc are much more similar in terms of importance. Furthermore, even the best business attribute predictor, `stars_business`, is one order of magnitude less important than the worst sentiment predictor, nrc.

We now assess accuracy.

```
[1] "Percentage Correct:  0.489679"
[1] "Percentage Wrong:  0.510321"
```

Given that this is essentially the bing only model, we are not surprised to get an accuracy score of roughly 49%.

3.3.4 Tuning Tree

We wanted to increase the size of the tree to see if that could improve accuracy. To do that, we dropped the default `cp` cutoff from 0.01 to 0.001, and refitted the tree, as seen below:

```
tuning <- rpart.control(cp = 0.001)

big.kitchen.sink.rpart <- rpart(stars_review ~ ., data = train.small, method = "class",
printcp(big.kitchen.sink.rpart)
```

```
Classification tree:
rpart(formula = stars_review ~ ., data = train.small, method = "class",
    control = tuning)

Variables actually used in tree construction:
[1] afinn_score    bing_score     stars_business

Root node error: 104058/183990 = 0.56556

n= 183990

         CP nsplit rel error  xerror      xstd
1 0.0512695      0   1.00000 1.00000 0.0020433
2 0.0340003      1   0.94873 0.94880 0.0020555
3 0.0039689      2   0.91473 0.91524 0.0020598
4 0.0032866      4   0.90679 0.90683 0.0020604
5 0.0024602      7   0.89693 0.89762 0.0020608
6 0.0020662      8   0.89447 0.89657 0.0020609
7 0.0014607      9   0.89241 0.89545 0.0020609
```

```
8 0.0011916      10    0.89095 0.89293 0.0020610
9 0.0010000      11    0.88975 0.89023 0.0020610
```

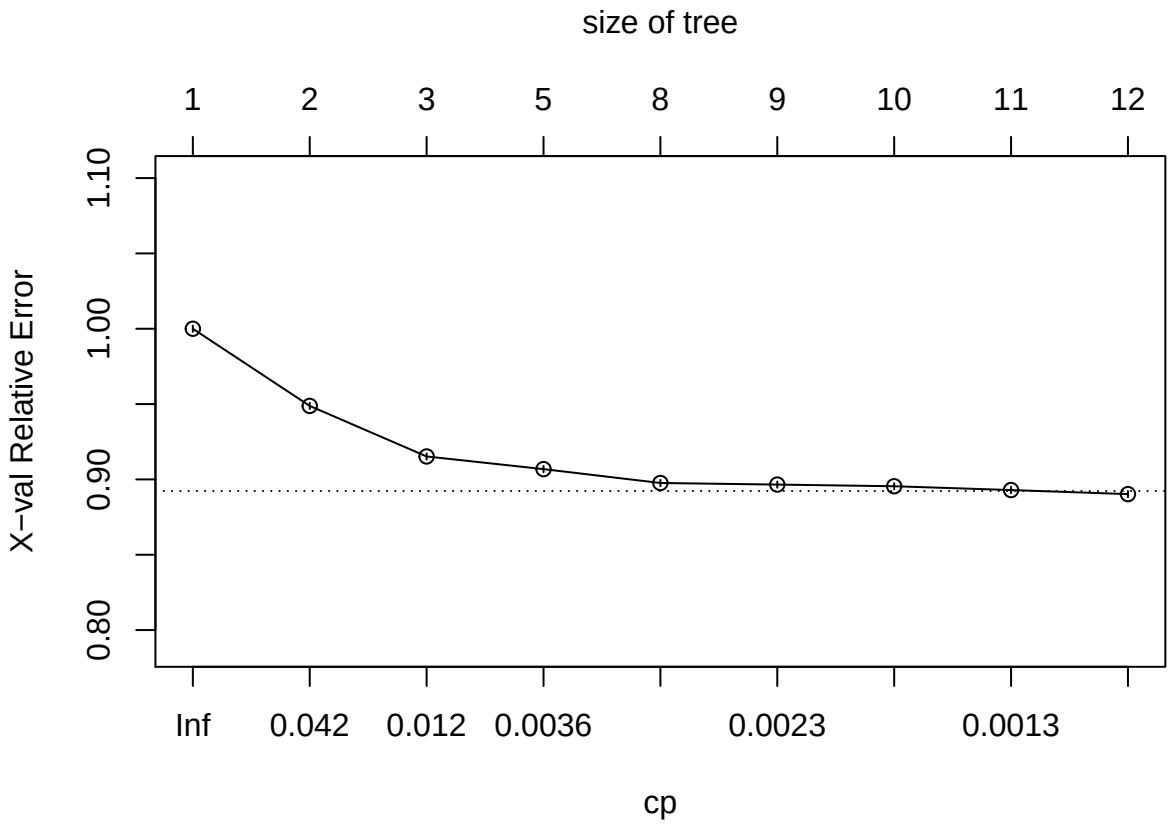
Unfortunately, the rpart.plot was too big to print, but we can see from the `printcp` summary that the tree used 8 variables in total, with the addition of AFINN, two ratings of the review by other users (`cool`, `useful`), and four business attributes (`postal_shrunk`, `stars_business`, `Sunday`, `Type`)

```
[1] "Percentage Correct:  0.503431"
[1] "Percentage Wrong:  0.496569"
```

However, even with the addition 7 new variables, we have only achieved a marginal improvement in performance, with an accuracy of roughly 50.0%. While this is definitely the highest accuracy, it barely improves on the bing only model accuracy of 49.2%

This conclusion is also reflected in the drop in complexity with each further split, as shown below.

```
plotcp(big.kitchen.sink.rpart)
```



We can see that by 7 splits, there is essentially no improvement in the accuracy of the model.

3.4 Random Forest

Extending the tree model, we hope that with a random forest its properties of variance reduction through averaging many deep trees, we can achieve a higher accuracy.

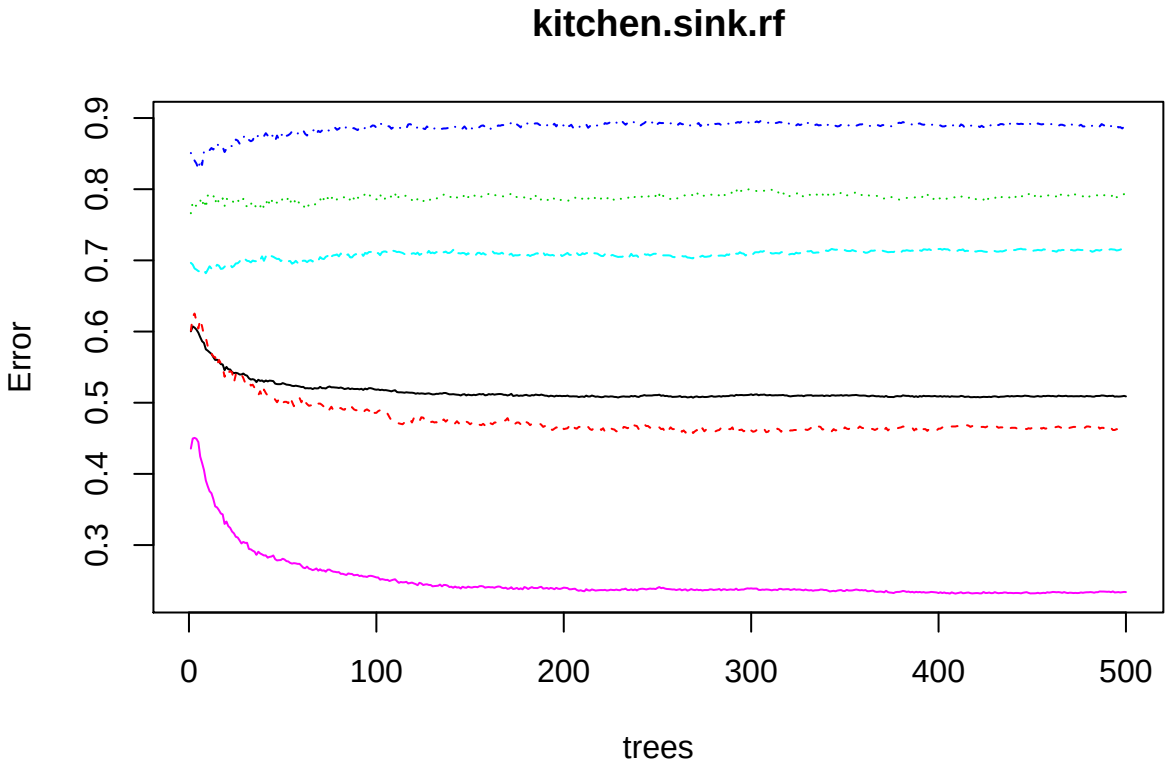
```
Call:
 randomForest(formula = stars_review ~ ., data = train.small,      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 6

        OOB estimate of  error rate: 50.89%
Confusion matrix:
    1   2   3    4    5 class.error
1 872 306 113  146  181   0.4610630
2 476 348 167  343  338   0.7918660
3 261 227 236  587  800   0.8882046
4 143 174 203 1417 3016   0.7139108
5 122 130 175 1455 6163   0.2339341
```

From the `class.error` in the Confusion matrix, we can see that the same problem with the Classification Trees exist, in that the lowest error predictions are for class 5, with an error rate of 23.4%, jumps a significant amount for class 1, with an error rate of 46.1%, and jumps again to class 4, with an error rate of 71.3%, and stabilizes for 2 and 3, with an error rate of 79.1% and 88.8% respectively.

We can further explore how the error rate for the randomForest changes by plotting the error rate for each class and the overall OOB error, or Out of Bag Error. Since randomForest samples a subset of the predictors for each tree, (in the case of classification trees, usually sqrt of the number of predictors), and trains the tree on a bootstrap sample with replacement of the dataset, there are observations left out that can be used as an in-built test set. This process approximates cross-validation, the current standard method for trying to measure the true error of a model.

A plot of the error rate for each class and overall OOB error rate is shown below. We know that the OOB error is the black line, as shown in reference [`http://statweb.stanford.edu/~jtaylo/courses/stats202/ensemble.html`][1]. We can infer from the ending class error what class each line represents. At the right side of the plots, from top to bottom, the lines are class 3, 2, 4, OOB, 1, and 5.



What's interesting is that with more trees, the error rate for class 3 actually increases, and for classes 2 and 4 stay roughly the same. All improvement in the OOB error comes from class 1, and most dramatically class 5. This further reinforces how trees fit very poorly to the intermediate values of 2-4, and the clear lack of correlation between the predictors and intermediate values 2-4.

Next, we measure accuracy.

```
[1] "Percentage Correct:  0.478203"
[1] "Percentage Wrong:  0.521797"
```

Disappointingly, we find that the Random Forest under-performs both the bing only model and kitchen sink tree model. This suggests that our problem with the classification trees is not over-fitting but rather poor predictors themselves.

3.4.1 Variable Importance

To examine variable importance, we decided to forgo the built in varImpPlot function, as with many variables it is difficult to read. We extracted variable importance measures manually and they are displayed as dataframe below:

```
Selecting by MeanDecreaseGini
```

	variables	MeanDecreaseAccuracy
1	bing_score	97.71905
2	afinn_score	75.33091
3	loughran_score	45.13696
4	postal_shrunk	33.70282
5	nrc_score	33.14230
6	useful	29.37888
7	Type	27.52486
8	latitude	26.18200
9	longitude	26.07931
10	review_count	24.61011

```
Selecting by MeanDecreaseGini
```

	variables	MeanDecreaseGini
1	afinn_score	1438.6365
2	bing_score	1415.5061
3	nrc_score	963.5356
4	postal_shrunk	700.4611
5	loughran_score	678.3631
6	review_count	542.1468
7	longitude	511.9169
8	latitude	507.1569
9	Type	466.5227
10	useful	448.3012

In both measures of variable importance, we see that the sentiment predictors are the most important, with bing and AFINN trading places at the top, and loughran and nrc within the top 5. The only business attribute predictor to stay within the top 5 is `postal_shrunk`, and both `latitude` and `longitude` make it in the top 10, showing the importance of location for reviews. The three remaining stragglers are review_count, Type, and useful.

3.4.2 Variable Impact

First, apologies for the placement of the plots, we tried multiple scaling arguments to get the plots in the right order with the text, but were not successful.

To examine the impact of these variables, we an use `partialPlot`, which show the relative logit contribution of the variable on the class probability from the model we fitted. In simple terms, this means that postie values on the y-axis means a higher likelihood for that value of the predictor, which is on the x-axis, with the opposite for negative values. The class being predicted can be adjusted by the `which.class` parameter. We do not examine `Type` as it is a categorical variable and does not play well with `partialPlot`.

First, we examine the variable `longtiude`. The first plot is trying to see the impact of `longitude` on the likelihood of review score 5, whereas the second plot is doing the same for review score 1. The code below is shown as an example, future analysis will have the code hidden.

```
longitude.5 <- partialPlot(kitchen.sink.rf, as.data.frame(test),
                           longitude, which.class = '5',
                           main = "Partial Dependence on Longitude for Review Score 5"
```

```
longitude.1 <- partialPlot(kitchen.sink.rf, as.data.frame(test),
                           longitude, which.class = '1',
                           main = "Partial Dependence on Longitude for Review Score 1"
```

Cleveland has a longitude of around -80, whereas Phoenix has a longitude of around -110. Thus, if forced to make a conclusion on the impact of geography on the review score, we could say that there is a slight increase in the likelihood of having a review score of 5 and a slight decrease of having a review score of 1 in Phoenix compared to Cleveland. However, the impact is small and noisy, and may be spurious.

Next, we examine the variable `useful`. The first plot is trying to see the impact of `useful` on the likelihood of review score 5, whereas the second plot is doing the same for review score 1.

What's most interesting about these plots is that the higher the number of users who think that a post is useful, the less likely it is to be predicted a 5, and the more likely it is to be predicted a 1. The intuition behind this could be that users find critical reviews more honest and trustworthy.

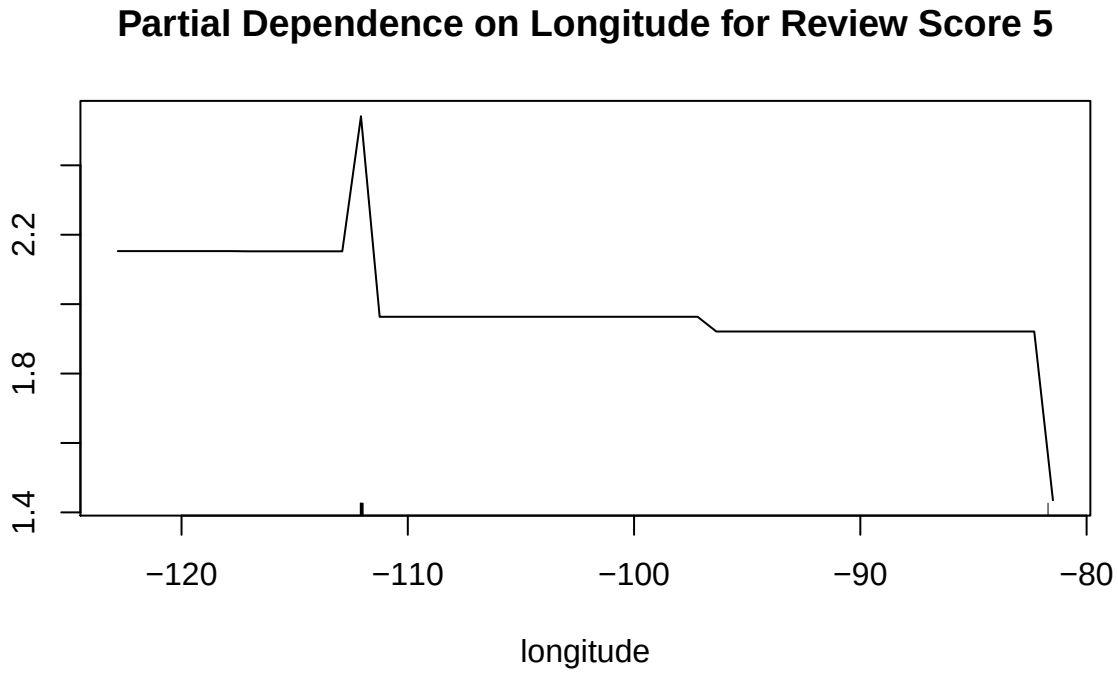


Figure 3.1: Partial Dependence on Longitude for Review Score 5

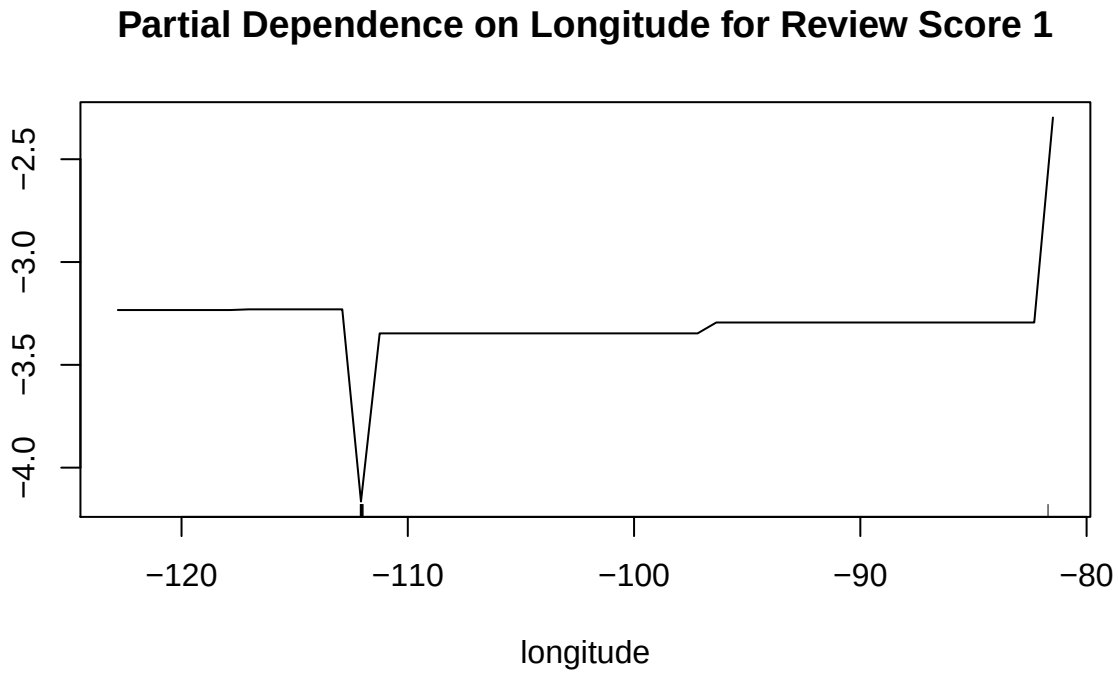


Figure 3.2: Partial Dependence on Longitude for Review Score 1

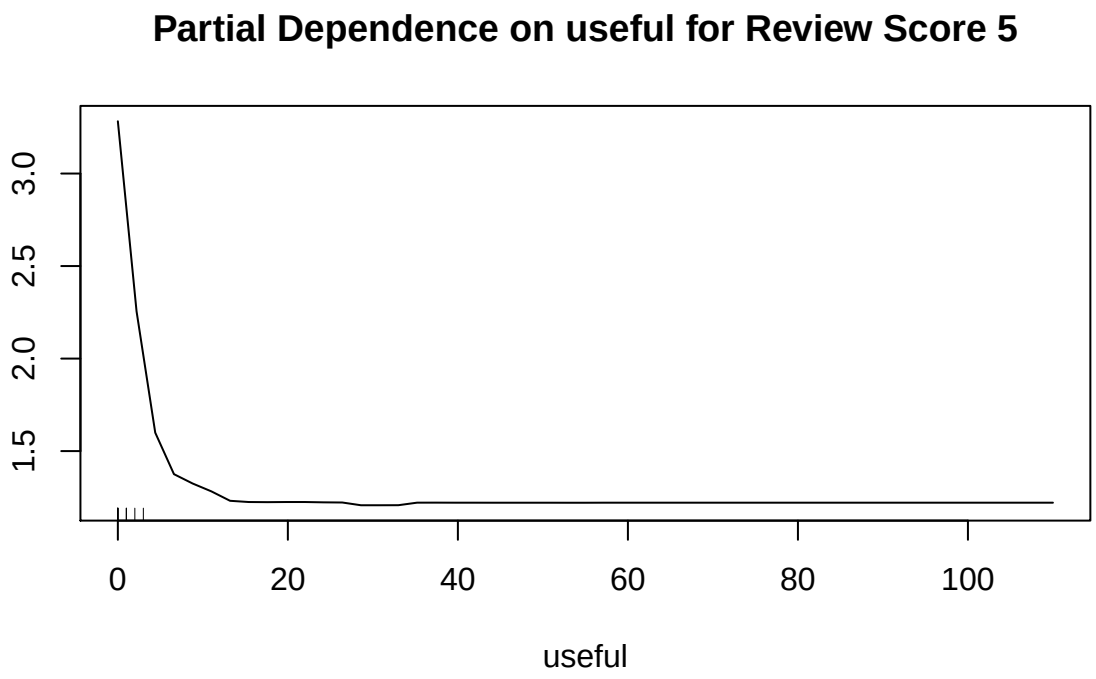


Figure 3.3: Partial Dependence on useful for Review Score 5

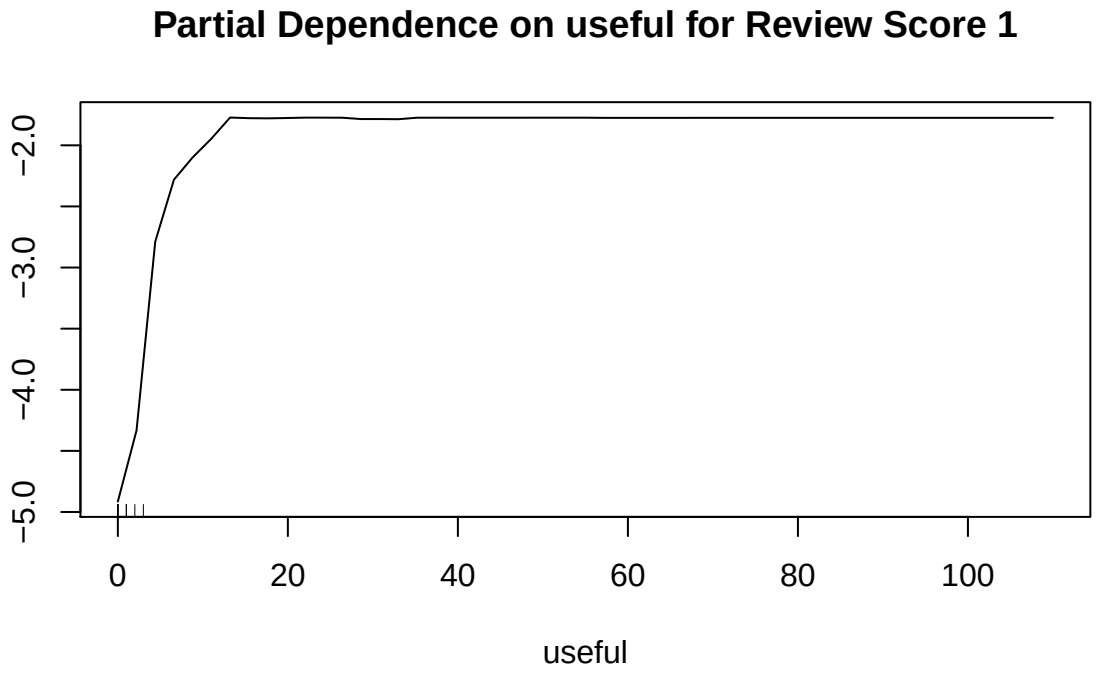


Figure 3.4: Partial Dependence on useful for Review Score 1

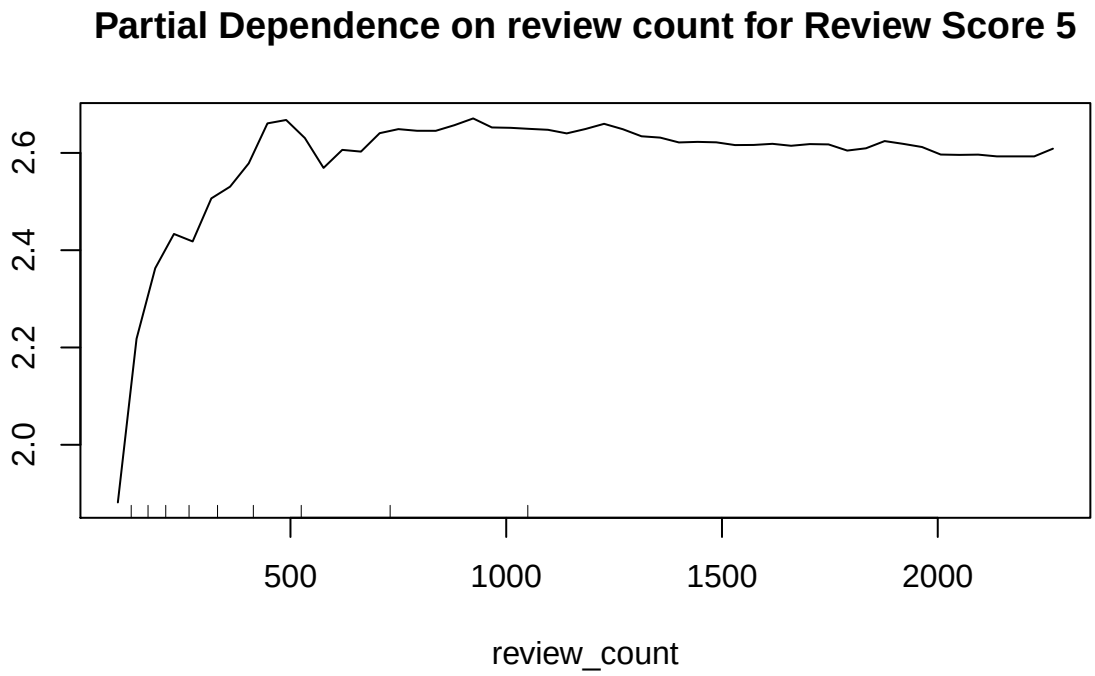


Figure 3.5: Partial Dependence on review count for Review Score 5

Finally, we examine the variable `review_count`. The first plot is trying to see the impact of `review_count` on the likelihood of review score 5, whereas the second plot is doing the same for review score 1.

The results from `review_count` make intuitive sense, as the first plot shows that more reviews increases the likelihood of a review score of 5, and the second plot shows that more reviews lead to a lower likelihood of being predicted class 1.

3.5 Best Model, Conclusion

Our best model, surprisingly, is essentially a close tie between the bing only tree, all lexicon tree, kitchen sink tree, and expanded kitchen sink tree, with an accuracy on the test set of 49.2%, 49.1%, 48.9%, and 50.3%. However, the expanded kitchen sink adds 7 variables with an increase in accuracy of roughly 1%. Moreover, the kitchen sink tree and the bing only tree are essentially the same model, with both only using the bing sentiment score in its predictions. The all lexicon tree expands this slightly with the inclusion the AFINN sentiment predictor, but has essentially the same accuracy. The Random Forest model, which usually is able to prevent over-fitting on the training set

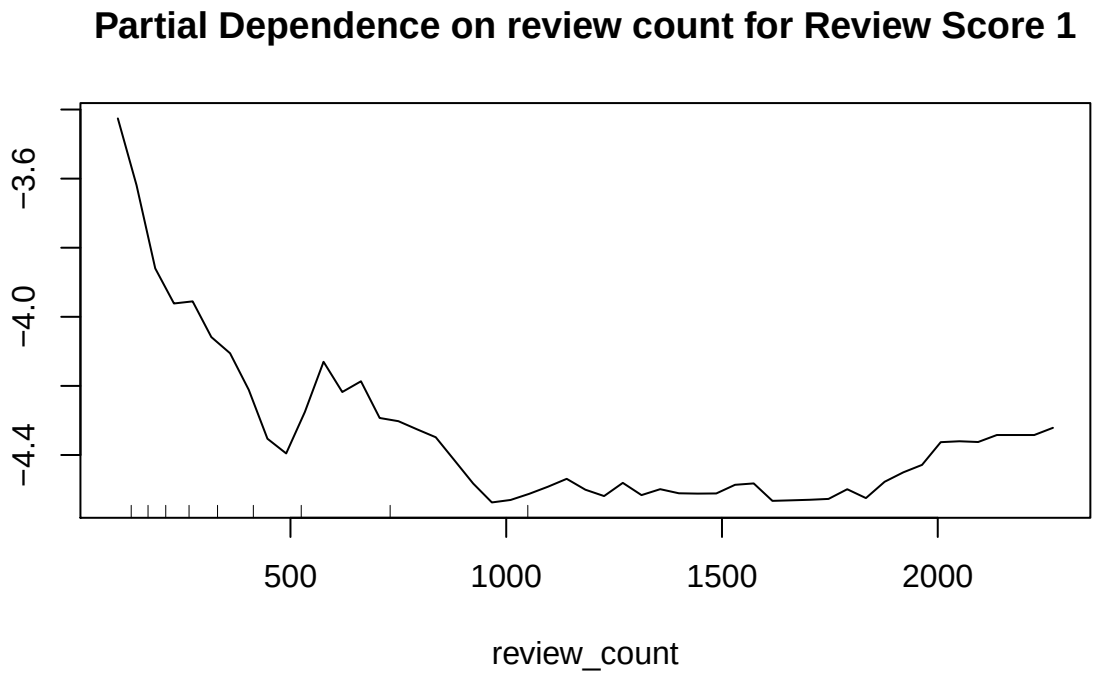


Figure 3.6: Partial Dependence on review count for Review Score 1

and has more stable predictions due to its variance reduction properties, only had a accuracy of 47.8% on the test set.

Since we have no true standout models here, we would argue that the best model would be a tree with only the bing sentiment predictor. This model improves over the baseline majority prediction of 45.5% by roughly 3.5%. This is not a substantial improvement, and goes to show how difficult it is to predict review scores into its specific categories. All of our predictors, including 20+ business attributes, fail to have any significant impact on review score. Finally, sentiment scores are only somewhat reliable when predicting extremes, i.e. either 1 or 5, and have extremely poor accuracy when it comes to any intermediate score, or 2-4. This goes show that sentiment is poorly related to intermediate review scores, and somewhat related to review scores of 1, and most strongly related to review scores of 5. However, this may just reflect the underlying distribution of review scores, which is highly skewed to a review score of 5.

For our final objective, to see if there were any differences between the East and West of the US, we can tentatively conclude that there are no significant differences. Except for the expanded tree and randomForest, location variables were ignored. Even when looking at the impact through marginal impact plots in the randomForest, we

find the effect to by tiny and possibly spurious.

Conclusion

Our response variable, `stars_review`, has 5 levels from 1-5. Thus, predicting `stars_review` is a multi-class classification problem. We used Classification Trees and, a natural extension, Random Forest models to address this problem. We believe that Classification Trees allowed us to most directly meet our objectives.

First, we can conclude that although sentiment can predict extreme values, or review scores of 1 or 5, with decent accuracy, it performs poorly for intermediate values, or review scores of 2, 3, or 5. However, any predictive power is only marginal, as our best models only performed roughly 3.5% better than a baseline majority prediction.

Second, we can conclude that bing is the best sentiment predictor, given that is was the most important variable and most used variable in all of our trees.

Third, we can tentatively conclude that there are no significant differences in review score between the East and West of the US. The impact we found was minor and possibly spurious. Furthermore, our sample was limited to one city on each coast, and thus cannot be generalized.

Overall, there are some major caveats to the conclusions we derive from the modelling. First, for the kitchen sink models, adjustments were made to the dataset due to technical troubles. This may impact the validity of our conclusions regarding the larger models. Second, we have already implemented a variety of filters to limit our dataset, including limiting to two cities, which also impacts the generalizabilty of our analysis.

Further extensions for the modelling could include multi-class logistic regression, or multi-class Adaboost. Another improvement could be made if the models took into account that `review_stars` is an ordinal variable, or that there is a clear ordering of review ratings, versus a general categorical variable that has no clear ordering.

There are a few limitations associated with our sentiment analysis. First, the lexicons used have limited words in their "dictionaries". In particular, slang words, words spelled incorrectly, or modifications of words (i.e. "delicioso" instead of "delicious")

are not all included in the lexicons. These lexicons also carry out **unigram-based** sentiment analysis, which means they only consider singular words at a time. Qualifiers ("this restaurant was **not** good") are not considered, and it is clear that connecting these words changes their meaning individually. In addition, with the exception of AFINN, the lexicons weight the words equally. In other words, a moderately-negative word like "bad" and a strongly-negative word like "repulsed" are assigned the same "weight" of sentiment. In reality, we would want words that express a greater degree of disapproval to be scored more negatively, similar to what AFINN tries to accomplish.

This analysis can provide useful feedback for restaurant owners and investors when trying to evaluate their customers. Moving forward, we would be interested in incorporating n-grams sentiment analysis as opposed to unigram sentiment analysis. N-grams explores the relationships between multiple words by tokenizing by 2 or more instead of by 1 (as is the case in unigram analysis). This method can tokenize by pairs of adjacent words rather than by individual ones. This would help us incorporate context which would likely improve the accuracy of the sentiment analysis.

References

4.1 Chapter 1

1. Link to the Yelp Round 12 Dataset: https://www.yelp.com/dataset/challenge

2. Nishida, Kan. "Working with JSON data in very simple way." *Learn data science*, Medium, 29 Mar. 2016: https://blog.exploratory.io/working-with-json-data-in-very-simple-way-ad7ebcc0bb89

3. Hadley's Comments: https://community.rstudio.com/t/dplyr-alternatives-to-rowwise/8071

4.2 Chapter 2

1. "Text Mining: Sentiment Analysis." *Hierarchical Cluster Analysis*, UC Business Analytics R Programming Guide

2. Robinson, David. "Does Sentiment Analysis Work?" *Variance Explained*.

3. Silge, Julia, and David Robinson. "Sentiment Analysis with Tidy Data." *Text Mining with R*, 23 Sept. 2018.

4.3 Chapter 3

1. Random Forests. (n.d.). Retrieved from http://statweb.stanford.edu/~jtaylo/courses/stats202/ensemble.html