CYO - Hotel Rating Prediction Project

Yu-Wei Hung

01/12/2020

Introduction

Travel, in this day and age, is prevalent and modernized with websites like tripadvisor, Trivago, Expedia, etc. These websites provide user generated reviews for hotels and resorts all around the world for potential traveler's reference. Specifically, the popular website TripAdvisor made revenue accumulating to 1.56 billion USD in 2019, a large sum of cash for a seemingly simple idea. The idea behind these websites are to provide potential travelers with honest feedback, concerns, suggestions, and recommendations made only by previous visitors to promote unbiased and honest reviews, as opposed to reviews and recommendations that can be bought off by large hotel corporations. Most of these websites go further in recommending hotels by collecting previous user reviews and preferences, and most also corporate with hotels to offer purchasing of rooms and suites at a discounted rate. Usually, visitors are able to rate their stay and experience on a 5-star system with an text box to complement the rating with further explanation as to why the resorts/hotels were given this rating. However, this rating system is not without flaws. Sometimes people would rate a place 4/5 even though they specify no problems or flaws with the place, or sometimes people may rate a place 5/5 even though they would provide several flaws and bad experiences from the place. In order to prevent this bias from influencing the local hotel/resort business, a algorithm that predicts or assigns a rating by keywords from the text box can help AI systems from better sorting and organizing reviews.

The main goal of this project is to build a prediction system that uses sentiment analysis based on keywords provided by users to predict the ratings that the specific review might be. Although extremely limited in essence, this project may provide insight to further recommendation systems or AI systems that can better rule out outliers or uncorrelated reviews.

In this document we first take organize the data set. Then, we explain the process of data exploration, visualization, and analysis. After, we present the prediction approach of this hotel prediction system and explain the usage, limitations, and potential this project entails.

Dataset

The data set that is used in this project is from Kaggle by Larxel, that has gathered a large amount of movie rating data, which is in the zenodo website, by Barkha Bansal.

The complete database can be found here: (https://raw.githubusercontent.com/yhung615/Hotel-Capstone/main/tripadvisor_hotel_reviews.csv)

The version of R used in this project and any subsequent code is for the version R 4.0.

Model Explanation

Model Loss Equations

Mean Squared Error (MSE)

$$MSE = \frac{1}{N}\sum_{i=1}(\hat{y}_i - y_i)^2$$

The mean squared error, or MSE, is a statistical estimator that measures the averages of squared errors. The value is always positive, and values approaching 0 are better estimators.

Mean Absolute Error (MAE)

$$MAE = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n}$$

The mean absolute error, or MAE, is a type of error that calculates the averages or the errors without consideration of direction.

Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}(\hat{y}_i - y_i)^2}$$

The root mean squared error, or RMSE, is the square root of MSE. It is the target metric we will use in evaluating our recommendation model.

Method/Analysis

In order to get started, we must download all the required packages that will be needed in this project.

```
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(dplyr))
  install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(tidytext))
  install.packages("tidytext", repos = "http://cran.us.r-project.org")
if(!require(wordcloud))
  install.packages("wordcloud", repos = "http://cran.us.r-project.org")
if(!require(wordcloud2))
  install.packages("wordcloud2", repos = "http://cran.us.r-project.org")
if(!require(RColorBrewer))
  install.packages("RColorBrewer", repos = "http://cran.us.r-project.org")
if(!require(tm))
  install.packages("tm", repos = "http://cran.us.r-project.org")
if(!require(rsample))
  install.packages("rsample", repos = "http://cran.us.r-project.org")
```

First the data is retrieved from a copied version on a github repository and processed using a comma separator.

```
dl <- tempfile()
download.file(
  "https://raw.githubusercontent.com/yhung615/Hotel-Capstone/main/tripadvisor_hotel_reviews.csv",dl)
rawdata <- read.csv(dl, sep = ",")
```

After the raw data has been extracted, we must organize the data so that we have all of the columns and rows where we needed it to be. String splitting is necessary in lengthy word reviews such as this dataset in order to make sure all words are taken in account. The complementing tokenization is also required at this stage for sentiment analyses. In the code below, Bing is used to retrieve sentiments, which are emotional values assigned to words and phrases. It can either be positive and negative and has a value assigned to it. There are other lexicons to use, but for the sake of this project, bing is selected.

```
rating <- rawdata %>%
  group_by(Review) %>%
  mutate(
    reviewNum = row_number(),
    vocab = str_split(Review, " ")) %>%
  ungroup() %>%
  unnest_tokens(word, Review)

data <- rawdata %>% rownames_to_column()


sentiment_data <- data  %>%
                  unnest_tokens(word,Review) %>%
                  anti_join(stop_words) %>%
                  inner_join(get_sentiments("bing"))%>% ##Retrieve sentiment values for keywords##
                  count(Rating ,index= rowname, sentiment)%>%
                  spread(sentiment, n, fill = 0) %>%
                  mutate(sentiment = positive-negative)   ##Generate overall sentiment values##
```

Now that the data has been processed, we can move onto data exploration and visualization.

General Statistics

Before we get started, lets take a look at the dataset:

```
str(sentiment_data)
```

```
## tibble [20,441 x 5] (S3: tbl_df/tbl/data.frame)
##  $ Rating   : int [1:20441] 1 1 1 1 1 1 1 1 1 1 ...
##  $ index    : chr [1:20441] "100" "10018" "10096" "10116" ...
##  $ negative : num [1:20441] 10 21 5 9 6 1 2 6 11 19 ...
##  $ positive : num [1:20441] 5 6 3 6 5 3 7 2 6 14 ...
##  $ sentiment: num [1:20441] -5 -15 -2 -3 -1 2 5 -4 -5 -5 ...
```

From this line of code, we can see that there are 20,441 keywords analyzed using 5 indicators: Rating, index, negative, positive, and sentiment. Some of these classes are different, but we'll get to that later.

```r
head(sentiment_data)
```

```
## # A tibble: 6 x 5
##   Rating index negative positive sentiment
##    <int> <chr>    <dbl>    <dbl>     <dbl>
## 1      1 100         10        5        -5
## 2      1 10018       21        6       -15
## 3      1 10096        5        3        -2
## 4      1 10116        9        6        -3
## 5      1 10118        6        5        -1
## 6      1 1012         1        3         2
```

From this, we can see that a rating of 1 is prescribed with some negative and positive value, and an overall sentiment value. In the first row, it is predictable that sentiment value is negative, since the rating is 1. However, it is not always the case. As you can see on line 6, an overall sentiment value of 2 is generated, which indicates that even though the reviewer gave the institution a 1-star rating, the reviewer used an overall positive words in his recommendations text box.

```r
dim(sentiment_data)
```

```
## [1] 20441     5
```

From this line of code, we can see clearly that there is 20441 rows of data and 5 columns of variables.

```r
n_distinct(sentiment_data$Rating)
```

```
## [1] 5
```

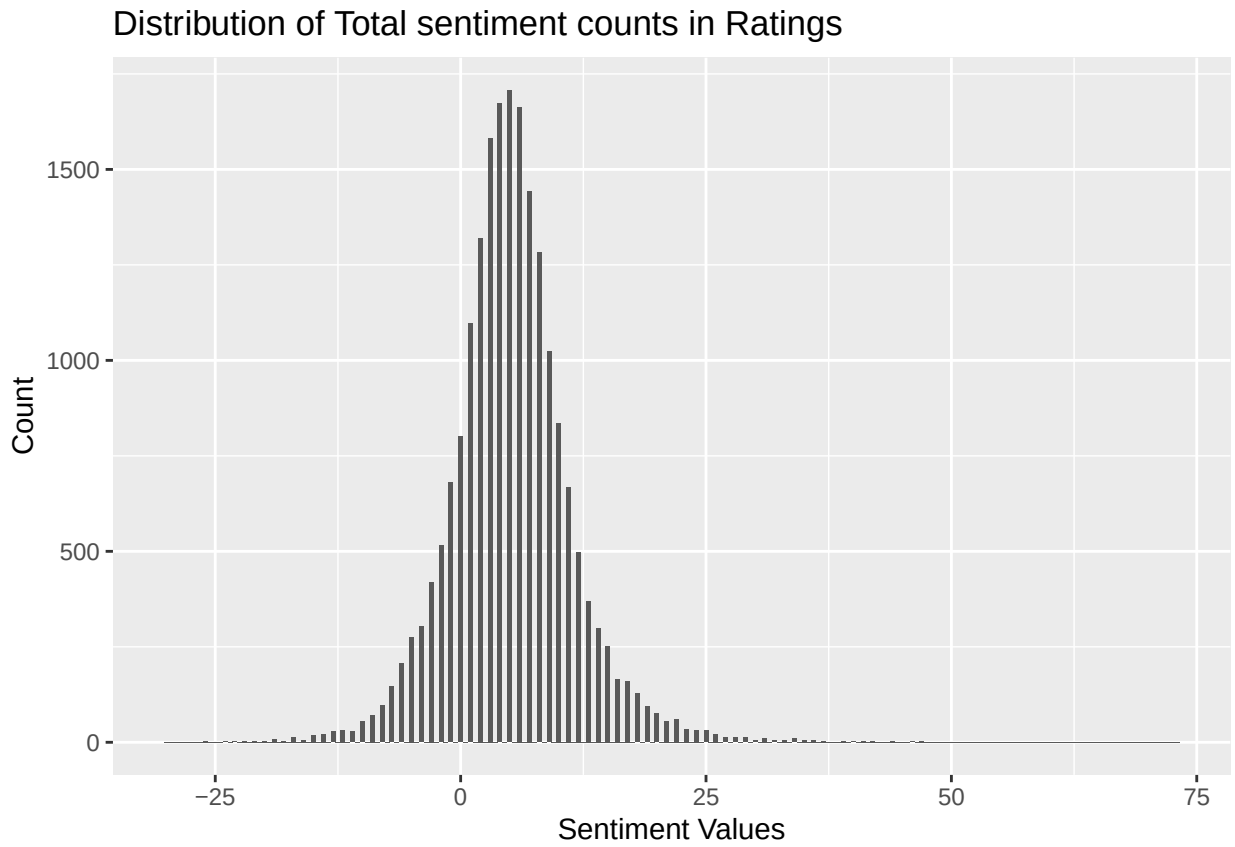
From this line of code, we can confirm that the ratings for this dataset is from 1 to 5, similar to other hotel websites 5-star rating system.

Data Visualization

Here we present data visualizations of the TripAdvisor Hotel Ratings dataset.

Sentiment Distribution

```r
sentiment_data %>% ggplot(aes(x=sentiment)) +
                      geom_bar(stat = "count",binwidth = .5)+
                      ggtitle("Distribution of Total sentiment counts in Ratings")+
                      ylab("Count") + xlab("Sentiment Values")
```



From this bar graph, the total distribution of sentiments in the entirety of the hotel ratings dataset is in the form of a normal distribution. This is expected, since we have a large amount of data and that tends to fall towards a normal distribution. An interesting point to note here is that the sentiment value is highest at around 0.5. From this, we can also take a look at the central tendency of our dataset.

```
mean(sentiment_data$sentiment)
```

```
## [1] 5.17959
```

```
median(sentiment_data$sentiment)
```

```
## [1] 5
```

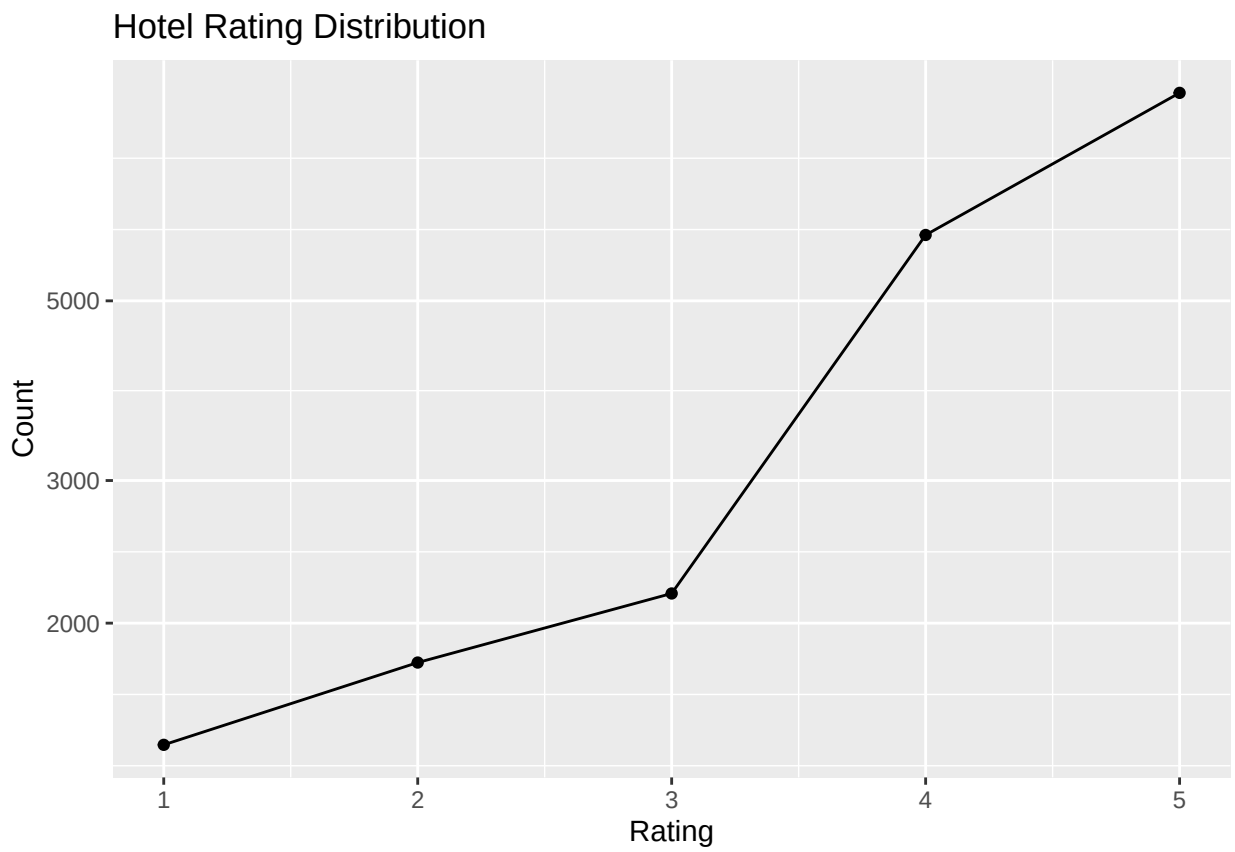
```
quantile(sentiment_data$sentiment)
```

```
##   0%  25%  50%  75% 100%
##  -30    2    5    8   73
```

From the mean, median, and quantile values, we can see that the average sentiment score lies around 5, while the max is 73 and min is -30.

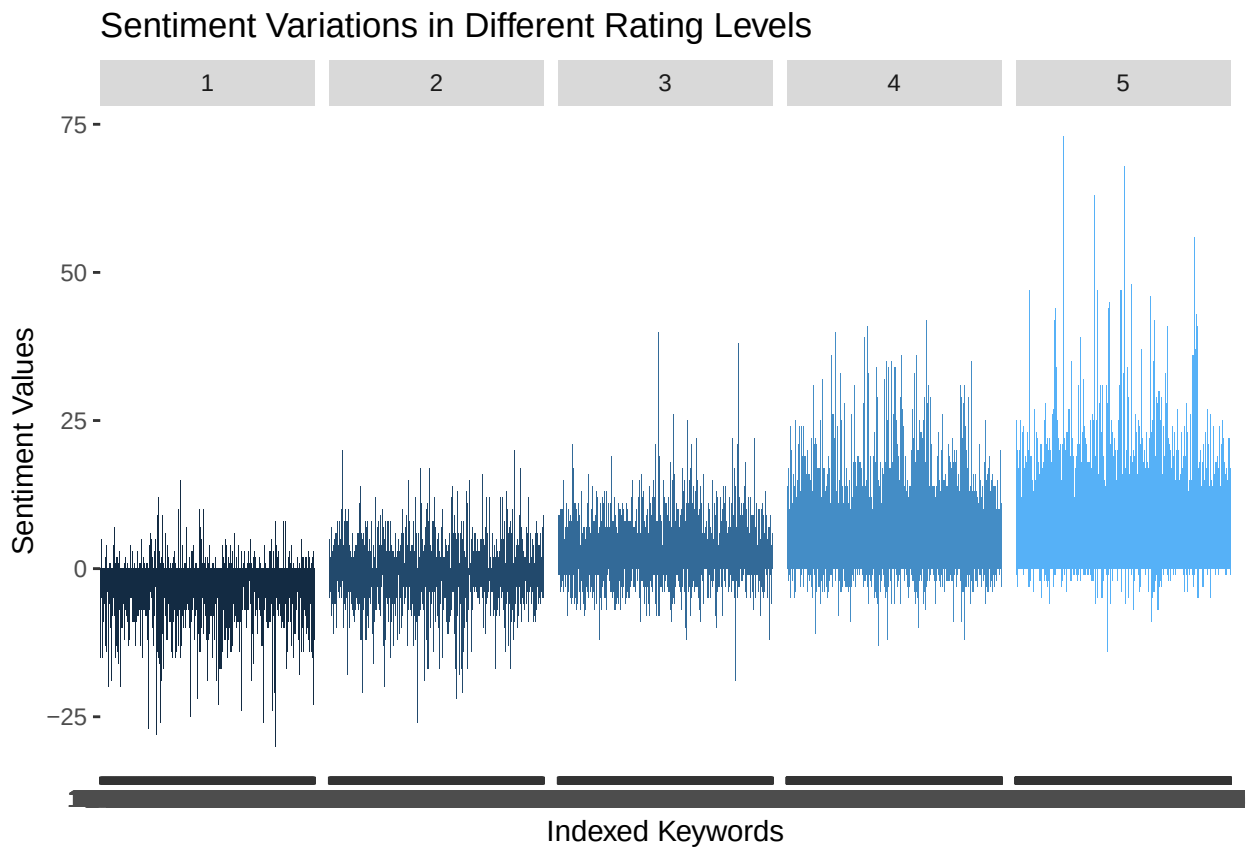
```
sentiment_data %>% group_by(Rating) %>% summarize(n = n()) %>%
                                                ggplot(aes(x = Rating, y = n)) +
                                                geom_point() + geom_line() +
```

```
                                                    scale_y_log10() +
                                                    ggtitle("Hotel Rating Distribution") +
                                                    ylab("Count") + xlab("Rating")
```



In this simple line chart, we demonstrate that the majority of the ratings are 5-stars and the minority are in the 1-star rating. There is higher propensity for ratings to be higher than lower.

```
sentiment_data %>%  ggplot(aes(index, sentiment, fill = Rating)) +
                    geom_col(show.legend = FALSE, width = 2) +
                    facet_wrap(.~Rating, ncol = 5, scales = "free_x") +
                    ggtitle("Sentiment Variations in Different Rating Levels") +
                    xlab("Indexed Keywords") + ylab("Sentiment Values")
```



From this graph, we can see a clear distribution of specific sentiment distributions of all rating levels. In 1-star rating, it is predominately negative, while in a 5-star rating, it is predominately positive. However, in 2-star and 3-star rating, there is a large mix of both negative and positive sentiment values. This distribution is expected, since lower ratings is correlated with lower sentiment values while higher ratings is correlated with higher sentiment values.

Specific Look on the relationship of positive and negative sentiment words

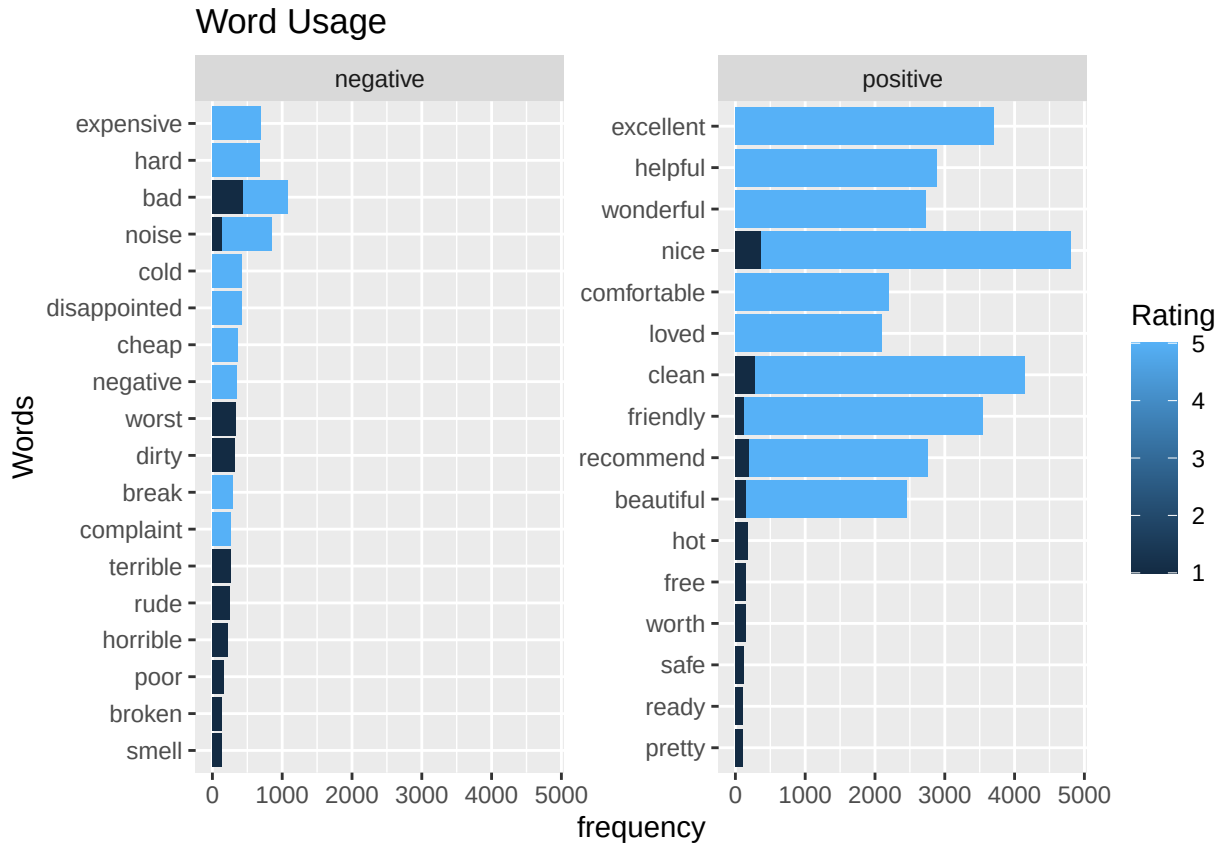
Although holistic view of sentiment data is necessary, it is also important to take a look at the dataset in regards to exactly what words are being used. Since reviews and recommendations are written by everyday users, it is important to explore this dataset with caution. For example, words like "not nice" may be gathered incorrectly as "nice" in the negative sentiment category, which is shown in the graph below. In this section, specific words used will be explored and visualized to get a better understanding of the datasets.

```
#We first generate an alternative sentiment data from our main dataset
#for alternative data visualization purposes##

alt_sentiment <- rawdata %>%
  filter(Rating %in% c(1,5)) %>%
  unnest_tokens(word, Review) %>%
  anti_join(stop_words) %>%
  inner_join(get_sentiments("bing"))

## A cumulation of popular words used for positive and negative
#sentiments is shown using this code below##
```

```
alt_sentiment %>% group_by(sentiment, Rating) %>%
  count(word) %>%
  top_n(10) %>%
  ggplot(., aes(reorder(word, n), n, fill = Rating)) +
  geom_col(show.legend = T) +
  coord_flip() +
  facet_wrap(~sentiment, scales = "free_y") +
  xlab("Words") +
  ylab("frequency") +
  ggtitle("Word Usage")
```



From this chart above, expected words such as “recommend”, “helpful”, “wonderful”, and “clean” is in the positive category, while “bad”, “terrible”, “worst”, and “dirty” is in the negative category. It is important to note the discrepancy of data for the negative category since as we have discovered above, the majority of sentiment falls in the positive category. Interesting observations can be seen in this chart above. The terms “expensive”, “disappointed”, “break”, “complaint” is in the negative sentiment category, but it is most often used for higher associated ratings. An explanation for this is that users might have used these specific words under different contexts. For example, the term “expensive” might be followed up with “worth”, or “valued”, while terms like “disappointed” and “complaint” might come after “no”, signalling that there’s “no complain”, or “not disappointed”. While possible explanations can formulate, more research must be conducted to discover why some words are found in different sentiment values and what biases might be associated with it.

Lets explore some of the words used in polar opposites (1-star and 5-star ratings systems). In order to achieve this, the alternative dataset created above is used.

```r
##Filtering out everything except for 1-star ratings##
worst_senti <- alt_sentiment %>% filter(Rating == 1)

##making sure everything is correct and acknowledging this alternate dataset's parameters##
str(worst_senti)
```

```
## 'data.frame':    17364 obs. of  3 variables:
##  $ Rating   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ word     : chr  "horrible" "appealing" "free" "fine" ...
##  $ sentiment: chr  "negative" "positive" "positive" "positive" ...
```

```r
##Processing alternate dataset for generating wordclouds##
##Load dataset into Corpus##
neg_doc <- VCorpus(VectorSource(worst_senti$word))

##Replace special characters##
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
neg_doc <- tm_map(neg_doc, toSpace, "/")
neg_doc <- tm_map(neg_doc, toSpace, "@")
neg_doc <- tm_map(neg_doc, toSpace, "\\|")

##Apply all as lower case##
neg_doc <- tm_map(neg_doc,content_transformer(tolower))

##Removing words that's irrevelant to save processing space##
neg_doc <- tm_map(neg_doc, removeNumbers)
neg_doc <- tm_map(neg_doc, removeWords, stopwords("English"))
neg_doc <- tm_map(neg_doc, removeWords,
                  c("hotel", "rooms", "room", "service", "staff", "resort", "location"))

##Removing punctuation, extra white spaces, and text stem##
neg_doc <- tm_map(neg_doc, removePunctuation)
neg_doc <- tm_map(neg_doc, stripWhitespace)
neg_doc <- tm_map(neg_doc, stemDocument)

##Turn text into matrix##
tdm <- TermDocumentMatrix(neg_doc)
m <- as.matrix(tdm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(b_word = names(v),freq=v)
head(d, 10) #Shows first 10 words & sentiments##
```

```
##              b_word freq
## bad             bad  468
## nice           nice  386
## worst         worst  343
## dirti         dirti  325
## clean         clean  279
## terribl     terribl  279
## rude           rude  253
## smell         smell  251
## recommend recommend  248
## horribl     horribl  226
```

This step is repeated for 5-star rating system.

```
 ##Filtering out everything except for 5-star ratings##
best_senti <- alt_sentiment %>% filter(Rating == 5)

##making sure everything is correct and acknowledging this alternate dataset's parameters##
str(best_senti)
```

```
## 'data.frame':    111765 obs. of  3 variables:
##  $ Rating   : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ word     : chr  "wonderful" "excellent" "friendly" "smells" ...
##  $ sentiment: chr  "positive" "positive" "positive" "negative" ...
```

```
##Processing alternate dataset for generating wordclouds##
##Load dataset into Corpus##
pos_doc <- VCorpus(VectorSource(best_senti$word))

##Replace special characters##
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
pos_doc <- tm_map(pos_doc, toSpace, "/")
pos_doc <- tm_map(pos_doc, toSpace, "@")
pos_doc <- tm_map(pos_doc, toSpace, "\\|")

##Apply all as lower case##
pos_doc <- tm_map(pos_doc,content_transformer(tolower))

##Removing words that's irrevelant to save processing space##
pos_doc <- tm_map(pos_doc, removeNumbers)
pos_doc <- tm_map(pos_doc, removeWords, stopwords("english"))
pos_doc <- tm_map(pos_doc, removeWords,
                  c("hotel", "rooms", "room", "service", "staff", "resort", "location"))

##Removing punctuation, extra white spaces, and text stem##
pos_doc <- tm_map(pos_doc, removePunctuation)
pos_doc <- tm_map(pos_doc, stripWhitespace)
pos_doc <- tm_map(pos_doc, stemDocument)

##Turn text into matrix##
tdm1 <- TermDocumentMatrix(pos_doc)
m1 <- as.matrix(tdm1)
v1 <- sort(rowSums(m1),decreasing=TRUE)
d1 <- data.frame(g_word = names(v1),freq=v1)
head(d1, 10) #Shows first 10 words & sentiments##
```

```
##              g_word freq
## nice           nice 4740
## love           love 4418
## excel         excel 4070
## clean         clean 3863
## recommend recommend 3430
## friend       friend 3412
## help           help 3282
## wonder       wonder 2829
```

```
## beauti       beauti 2636
## comfort     comfort 2404
```

Next step is to create the word clouds for both 1-star ratings and 5-star ratings to display what are the most commonly used words of these two extremes of the rating system.

```
set.seed(2222)
 wordcloud(words = d$b_word, freq = d$freq, min.freq = 1,
           max.words=100, random.order=FALSE, rot.per=0.35,
           colors=brewer.pal(8, "Dark2"))
```

The word cloud above shows the most common words used in the 1-star system reviews. Expected words include “bad”, “worst”, “disappoint”, “rude”, “smell”, and “horrible’. Some unexpected words are”clean“,”nice“, and”recommend", but as explained above, words may be used under different contexts.

```
set.seed(3333)
 wordcloud(words = d1$g_word, freq = d1$freq, min.freq = 1,
           max.words=100, random.order=FALSE, rot.per=0.35,
           colors=brewer.pal(8, "Dark2"))
```

The word cloud above shows the most common words used in the 5-star system reviews. Expected words include “nice”, “love”, “clean”, “excel”, “recommend”, and "perfect’. Evidently, there are less unexpected words for positive sentiments, since positive words are harder to use under different contexts.

Model Preparation

Below are some codes to prep the dataset for model building.

```
##We drop the index to optimize for modelling algorithms##
drop <- c("index")
sentiment_data <- sentiment_data[,!(names(sentiment_data) %in% drop)]
```

Dataset is partitioned as below:

```
##Modelling Approach##

set.seed(1234, sample.kind="Rounding")
test_index <- initial_split(sentiment_data, prop = 0.7,
                                    strata = "Rating")
train <- training(test_index)
test <- testing(test_index)

fitctrl <- trainControl(method = "cv", number = 10)
```

The reason why initial split was used instead of createDataPartition is because ranked/rating datasets is more suited for stratified split rather than a simple split. A proportion of 70/30 is used as this is a larger dataset to promote better processing time between training and testing data. ## Cross-Validation C5.0

First of the model building includes the C5.0 classification model with cross validation model building.

```
train_c5 <- train(as.factor(Rating) ~., data = train, method = "C5.0", trControl = fitctrl)

pred_c5 <- predict(train_c5, test)

####
#testing difference
diff <- (test$Rating)-(as.integer(pred_c5))
#Mean Squared Error
mse <- mean((diff)^2)
print(mse)
```

```
## [1] 1.145816
```

```
#Mean absolute error
mae <- mean(abs(diff))
print(mae)
```

```
## [1] 0.6763986
```

```
#Root Mean Squared Error
rmse <- sqrt(mse)
print(rmse)
```

```
## [1] 1.070428
```

The RMSE values resulted in 1.070428 and MSE value is 0.6763986.

C5.0 Random Forest

We performed another model building using C5.0 classification model with classification tree random forest modelling.

```
c5_rf <- train(as.factor(Rating)~., data = train, method = "rf", trControl = fitctrl)
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

```
c5_rf_pred <- predict(c5_rf, test)

####
#testing
diff <- (test$Rating)-(as.integer(c5_rf_pred))
#Mean Squared Error
mse <- mean((diff)^2)
print(mse)
```

```
## [1] 1.176154
```

```
#Mean absolute error
mae <- mean(abs(diff))
print(mae)
```

```
## [1] 0.6907519
```

```
#Root Mean Squared Error
rmse <- sqrt(mse)
print(rmse)
```

```
## [1] 1.084506
```

The results of RMSE is 1.084506 AND MSE resulted in 0.6907519. The RMSE value is actually higher than cross-validation methods used above. Lets test out other model approaches.

SVMLinear2 Model

In this approach, we used SVM (Support Vector Machine) method meant for classification and regressions.

```
svm_linear <- train(as.factor(Rating)~., data = train, method = "svmLinear2", trControl = fitctrl)

svm_pred <- predict(svm_linear, test)

####
#testing
diff <- (test$Rating)-(as.integer(svm_pred))
#Mean Squared Error
mse <- mean((diff)^2)
print(mse)
```

```
## [1] 1.408906
```

```
#Mean absolute error
mae <- mean(abs(diff))
print(mae)
```

```
## [1] 0.7447398
```

```
#Root Mean Squared Error
rmse <- sqrt(mse)
print(rmse)
```

```
## [1] 1.186973
```

The result of RMSE is 1.186973 and for MSE is 0.74473. The resulting RMSE is the highest of the model approaches indicating that SVMLinear2 does not do well with this dataset.

K Nearest Neighbor Model

In this approach, we use KNN, which is a method that approximates association from between variables and outcomes by averaging observations within vicinity.

```
knn <- train(as.factor(Rating)~.,
             data = train,
             method = "knn",
             tuneGrid = data.frame(k = seq(1,5,2)))

knn$bestTune
```

```
##   k
## 3 5
```

```
knn$finalModel
```

```
## 5-nearest neighbor model
## Training set outcome distribution:
##
##    1    2    3    4    5
##  990 1275 1513 4216 6316
```

```
knn_pred <- predict(knn, test)

#testing
diff <- (test$Rating)-(as.integer(knn_pred))
#Mean Squared Error
mse <- mean((diff)^2)
print(mse)
```

```
## [1] 1.16229
```

```
#Mean absolute error
mae <- mean(abs(diff))
print(mae)
```

```
## [1] 0.6876529
```

```
#Root Mean Squared Error
rmse <- sqrt(mse)
print(rmse)
```

```
## [1] 1.078096
```

```
###
```

The result of RMSE is 1.101193 and MSE resulted in 0.7032053. The resulting RMSE is the lowest of the prior approaches and appropriate for a larger dataset.

Fitting Linear Model

In this method, we utilize the linear regression model to perform lm model training. The Lm approach carries out regression by analyzing single stratum variance and co-variance.

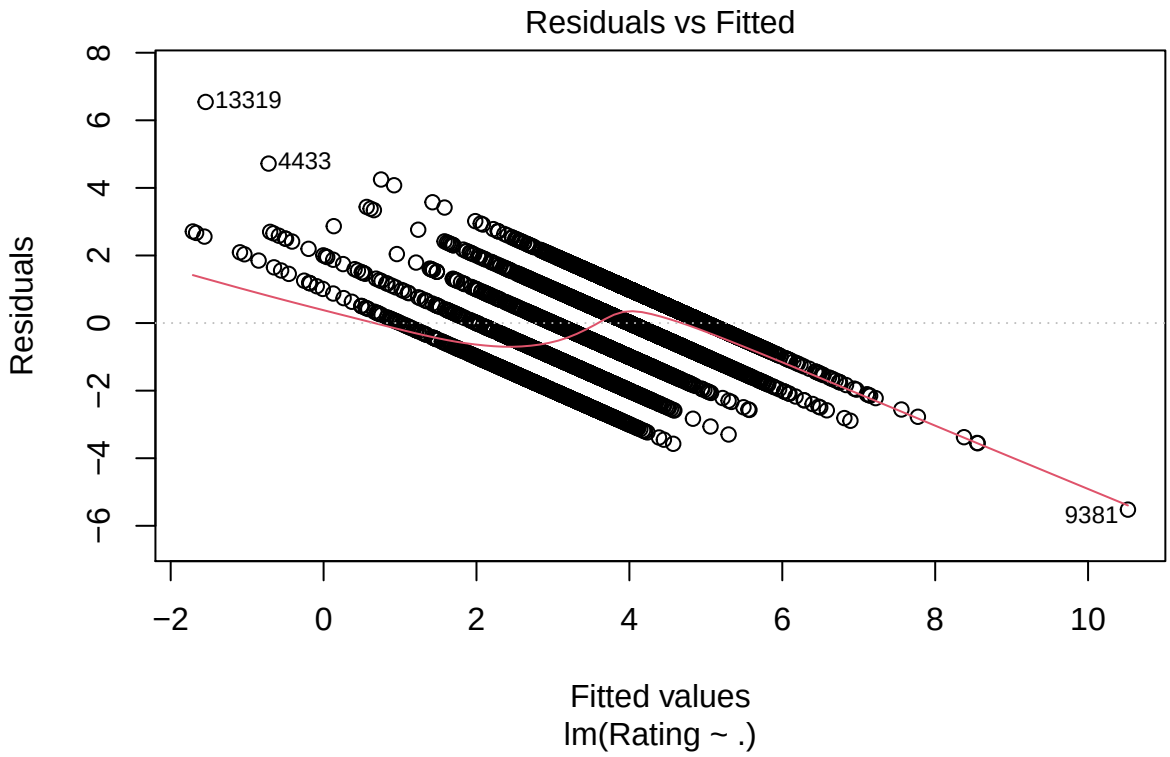
```
#Regression model training
#linear regression model is implemented using the lm() function.
#Rating is the dependent variable in the model and all other variables are used as predictors.
linear_reg <- lm(Rating ~. , data = train[-4])

#summary of the model
print(summary(linear_reg))
```

```
##
## Call:
## lm(formula = Rating ~ ., data = train[-4])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.5215 -0.5805  0.1397  0.7352  6.5431
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.712213   0.013340  278.29   <2e-16 ***
## negative    -0.157810   0.001887  -83.65   <2e-16 ***
## positive     0.093277   0.001320   70.69   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9849 on 14307 degrees of freedom
## Multiple R-squared:  0.3634, Adjusted R-squared:  0.3634
## F-statistic:  4084 on 2 and 14307 DF,  p-value: < 2.2e-16
```

From the summary above, its clear that regression model is significant (because its p-value is less than 0.05). The independent variables with name "positive"(p-value: 2e-16) and "negative"(p-value: 2e-16) are also significant.

```
##plot Residual and fitted.
plot(linear_reg,1)
```




```
##we can tell that the distribution on this chart is not clearly distributed.
```

```
#Prediction of model
pred_lm <- predict(linear_reg,test,type = "response")

#testing
diff <- test$Rating - pred_lm
#Mean Squared Error
mse <- mean((diff)^2)
print(mse)
```

```
## [1] 0.9489401
```

```
#Mean absolute error
mae <- mean(abs(diff))
print(mae)
```

```
## [1] 0.7788344
```

```
#Root Mean Squared Error
rmse <- sqrt(mse)
print(rmse)
```

```
## [1] 0.9741355
```

By using the lm approach, we arrived at a lowest RMSE value of 0.9734329 and a MSE value of 0.7775655. By trying out multiple approaches, we can conclude that the lm approach has the highest prediction accuracy rating.

```
print(test[10,1])
```

```
## # A tibble: 1 x 1
##   Rating
##    <int>
## 1      1
```

```
print(pred_lm[10])
```

```
##       10
## 3.611894
```

```
print(test[980,1])
```

```
## # A tibble: 1 x 1
##   Rating
##    <int>
## 1      3
```

```
print(pred_lm[980])
```

```
##      980
## 3.325019
```

Above are 2 examples of classification: the first example has predictions that are far from accurate, while the second example has predictions approaching on accurate.

Validation

Here we perform validation using the lm approach on the processed sentiment data instead of the testing data. Two different type of validation is performed, one with 10-fold cross-validation and one with 10-fold, 5-times cross validation.

```
set.seed(321)
train.control <- trainControl(method = "cv", number=10)
model <- train(Rating~., data = sentiment_data, method = "lm", trControl = train.control)
print(model)
```

```
## Linear Regression
##
## 20441 samples
##     3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
```

```
## Summary of sample sizes: 18397, 18397, 18397, 18397, 18396, 18396, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9814987  0.3663299  0.7830406
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
set.seed(321)
train.control <- trainControl(method = "repeatedcv", number=10, repeats = 5)
model <- train(Rating~., data = sentiment_data, method = "lm", trControl = train.control)
print(model)
```

```
## Linear Regression
##
## 20441 samples
##     3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 18397, 18397, 18397, 18397, 18396, 18396, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9816569  0.3661895  0.7831001
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
####
```

Under normal circumstances, 10-fold multiple times cross validation will result in a lower RMSE. However, in this case, the RMSE resulted lower for the 1-times 10-fold cross validation than the 3-times 10-fold cross validation.

Conclusion

Throughout the project, we've observed interesting statistics through our data visualization. However, the model prediction building ended up with a flawed prediction system to accurately predict ratings from sentiment values. Reasoning behind this flawed prediction system could be in the lack of a comprehensive model training and testing system, which requires a higher manipulation of the specific keywords for sentiment analysis. This being said, a lot of bias can result in a flawed prediction system. In the data visualization portion, many positively sentimented value are being used in lower ratings, which may result in the flawed system. In addition, the majority of sentiments from this database are postie values, and little are from the negative values, which makes an accurate prediction system for lower ratings more inaccurate than those in the higher ratings. Further testing is needed, but I hypothesize that this is the case. I believe further work can be done to separate different rating levels or even remove these ambiguous and contradicting words to make sure the prediction runs smoother.

However, this is not the permanent solution to hotel sentiment prediction systems, as these websites will continue to receive user recommendations that are not edited or vetted. This will result in persistent bias in the system, and further analysis and model building to counter that effect will be required for the prediction system to achieve a higher prediction rating.

Some of the limitations of the work include using datasets from only TripAdvisor's database as well as computing speed. Further work can be done to utilize multiple databases, as well as multiple lexicons to make sure we reduce these biases.

Although the prediction model was not as accurate as predicted due to many biases that still exists, the project serves to raise the potential of a keyword prediction system to help better the rating systems of not just hotels, but many other businesses and institutions to create a more accurate and unbiased rating system by using recommendation keywords to match specific ratings than to just allow users to select from an ambiguous 5-star rating system.