Do users' ratings reflect their true emotions?

Sentiment analysis of Yelp and TripAdvisor reviews to predict hotel ratings

Adam Post, Tianqi Wang, Ruoning Wang

ISA 414-B: Managing Big Data

Dr. Zhe Shan

10 May 2019

Abstract

We found statistical evidence to support the hypothesis that Yelp users with more emotionally-charged reviews will yield higher ratings on average. We fit a boosted tree model predicting review ratings for hotels with over 90% accuracy. In this model, the positive and negative sentiment-intensity scores of a particular textual review were used to predict the resulting review rating. Since we uncovered cointegration between Yelp and TripAdvisor hotel ratings over time, we trained our model with Yelp reviews and tested against TripAdvisor, predicting unavailable review ratings from their own sentiment-intensity scores. While we also considered time of post and hotel location as potential features, these variables lacked the predictive power of user sentiment.

Companies like TripAdvisor may use our model to predict how users really feel about their experience in a hotel. Through web scraping reviews related to their hotels, companies can run reviews through our model to gain customer insight. If model-predicted ratings are higher than actual hotel ratings, their particular customer base may have more rigid expectations than those of Yelp. Likewise, a model that under-predicts a user's hotel ratings may indicate more forgivable customers.

Introduction

With the sharp rise in Internet usage to facilitate commerce, online users are more likely to use the Internet to share their thoughts of products or services. As a result, review forums have played an increasingly important role as a determinant of demand. Quantifying the sentiments of online reviews can facilitate insight and provide managerial strategies. According to the Pew Research Center, "customers are often willing to pay 20% to even 99% more to buy a product of 5-star rate than that of 4-star rate" (Horrigan). Indeed, extracting meaningful information from any text review is not just an exercise in data mining; industries value greatly from predicting users' opinions, given the intensity and valence of their reviews. For example, many European hotels allow their customers to write a review before checking out. These unrated reviews can be extremely time-consuming to parse. If a star rating can be assigned to a review, the ratings can be averaged to assess hotel performance without explicitly requiring a rating from the customer.

A powerful consequence of user sentiments' strong predictive power is the outlier analysis that may be performed. Businesses may web scrape online review sites and compare ratings against sentiment scores, while looking for outlier customers that may feel a certain way towards a product while rating it differently. This information could be leveraged to identify and better target these users.

To illustrate this business problem, we analyzed textual hotel reviews from Yelp and found a strong relationship between a user's written perception of a hotel and their final rating, which we modeled through a boosted tree, which yielded an out-of-sample r^2 of 0.7697. However, Yelp's final ratings are discrete, meaning that hotels with varying sentiments may still be rated 4-stars without differentiation. This means that our sentiment scores were instead available, customers could make more informed purchasing decisions, in turn adding more credibility to the review site.

In terms of data analytics, we predicted numeric Yelp ratings against numeric polarity scores for each textual review, along with a singular-value decomposed document-term matrix from Yelp reviews. While we also incorporated geographic location and time as predictors, they did not explain nearly as much variability. In addition, we examined the reviews of TripAdvisor and ran the same model to predict ratings for TripAdvisor reviews, which were not available in the data set. We believe these predictions will maintain the same high level of accuracy for TripAdvisor as for Yelp, since we explored hotel reviews that overlapped in location and time and found that their time series are cointegrated. Under the assumption that this trend occurs for all reviews across time, our model has the potential to be scalable to other online hotel review sites.

Data Collection

Our first data source comes from https://www.yelp.com/dataset/challenge. As part of Yelp's dataset challenge, they have offered a large subset of review, business, and user information. The original review.json file contains 6,685,900 text reviews stored as JSON objects, along with review dates, corresponding ratings, business names, business types, and other information. These reviews contain business IDs that serve as primary keys linking to a business.json file containing 192,609 businesses, also stored as JSON objects. This collection contains business names, business locations, review counts, and other information.

While both of these files encapsulate many types of businesses, we are interested in hotels for our analysis, especially since TripAdvisor data only contains hotel information. Before joining original review.json and business.json, we took a subset of all businesses related to hotels using a regular expression. Then, we subsetted variables of interest—namely, the business ID, business name, city, state, review count, and rating, quantified by stars. We also subsetted variables of interest from review.json—namely, business ID, review date, rating, the text review itself, and counts of user metrics pertaining to the review (e.g. "cool", "funny", and "useful"). From there, we performed an inner join to merge the two data sets by business ID to complete our final Yelp data set, which contains 343,476 unique observations.

Our second data source comes from http://kavita-ganesan.com/entity-ranking-data/#.XLytxZNKjOQ. Data scientist Kavita Ganesan compiled approximately 259,000 hotel reviews in Dubai, Beijing, London, New York City, New Delhi, San Francisco, Shanghai, Montreal, Las Vegas, and Chicago, with approximately 80-700 hotels in each city.

The underlying data management was especially poor for this data set, requiring much more preparation than Yelp's data set, despite significantly lower data volume. The *data* folder from the TripAdvisor .zip has ten .csvs, where each .csv contains hotel names, addresses, review counts, and ratings for a particular city, along with a document ID serving as the primary key. The *data* folder also contains ten folders, each representing a city. For some reason, these folders hold reviews for each hotel contained as a separate file. Even worse, these files have no file extension!

This anomaly in data collection resulted in us having to construct a loop within each folder, iterating through each file and adding ".txt". We repeated this process for each of the ten cities in the data set. This was only the beginning of our problems since we then had to construct a nested for loop that appended each set of hotel reviews within a particular folder, as well as for all folders in the collection. After removing reviews that were empty, or that contained unreadable characters such as emojis, our result was 220,336 hotel reviews in ten cities merged within one data frame. Since file names corresponded to hotel names, we appended the file name to the data frame and deleted the extension. Then, we extracted the review date, title, the review itself, and the hotel name as a primary key.

We also merged the metadata, combining the ten .csvs into one master file. At this point, we had two fully-merged data frames, both for hotel reviews and the hotels themselves. We performed an inner join, just as we did for the Yelp data set, merging hotel information with review information. As a final cleaning measure, we replaced “-1”s with states (for example, Beijing is a city, but we made it also a state to group objects easier). Since there are instances of hotel reviews without hotel metadata (for some reason), our final data frame contains 212,328 unique hotel reviews.

Data Preparation

In order to prepare our datasets to be used for prediction, we did some further cleaning on the two datasets in Python. We cleaned the two datasets in a consistent manner so that we can use our prediction model trained from the Yelp dataset to predict individual review ratings for the TripAdvisor dataset.

The first thing we did to our data was eliminate contractions. Words like “can’t”, “won’t”, and “I’ve”, were restored to their original compound verbs. This process, although minor, is important to our sentiment analysis. Removing punctuation marks in our next step would eliminated apostrophes and render these important verbs unidentifiable. Upon creation of our document term matrix, the Vader lexicon would fail to assign sentiment scores to these words. Fortunately, returning contracted words to their original form can be easily done with the help of a regular expression. Refer to our Python code (Sec 3.1 De-contracting) for more details.

Following the typical text cleaning process, we removed all stop words and punctuation marks. When we removed the punctuations, we actually removed all symbols that are not alphabetic. Apart from period and commas, symbols like dashes and backslashes were also removed. Note that when we removed parentheses in the reviews, we did not remove the text enclosed in the parentheses. We think there might be additional information contained in parentheses that may help explain user sentiment.

Removing the stop words and punctuation marks resulted around ten-thousand blank observations, probably because there were some reviews comprised solely of stop words or punctuation marks. Because these blank rows add no additional information, we removed these blank rows before build our document-term matrix. Luckily, this step was easily performed by filtering out the reviews that have a length of 0. For further information, refer to our Python code.

The most important step in our text cleaning processes is language detection. Because people around the world use Yelp, some people wrote reviews in French, Spanish, or even Chinese and Japanese. We decided to remove these reviews from our dataset, since the Vader lexicon cannot reliably handle words in other languages. If we want to analyze these reviews, we would have to use another lexicon or method. We first used the Google Language detection engine to determine the language of each individual review. However, this Python package was not able to distinguish between English and languages that do not use alphabetic characters (such as Chinese and Japanese). Therefore, we provided a regular expression to filter out all reviews that were not written in ASCII characters. Refer to section 3.4 of our Python code for more.

During our initial exploration of the data, we found out some people spelled “amazing” as “aaaaamazing” or “finally” as “finalllllly”. These intentionally misspelled words will significantly increase the sparsity of our document term matrix (we will have more columns). For the sake of efficiency, we decided to return these words to their correct forms (i.e “aaaaamazing” to “amazing”). Because English words have a maximum of two repeated characters, this step can be done without affecting correctly-spelled words.

The last thing we did before creating our document term matrix was to lemmatize the text, where we reduced every word in a sentence to its root noun form. According to Hafsa Jabeen of DataCamp, “Lemmatization, unlike Stemming, reduces the inflected words properly, ensuring that the root word belongs to the language” (Jabeen). In other words, it reduces a word to its grammatically correct root form. For example, if we have the word “flying”, stemming results in “fli”, which is not correctly-spelled. Lemmatization solves this issue by reducing “flying” to “fly”. Lemmatization can be instantly performed with the Textblob package. This python package performs text cleaning processes like lemmatization and spelling correction much faster than NLTK package. This step can be found in Section 3.6 of our Python code.

After all these preliminary text cleaning processes, we created a Normalized TF-IDF document term matrix. We decided to use Normalized TF-IDF since it standardizes the values of the resulting matrix and it imposes a penalty on words that are frequent in the text. Because we are building a prediction model, we want all the predictors to be normalized (scaled 0-1) since this is a requirement to implement a boosted tree.

Data Analytics

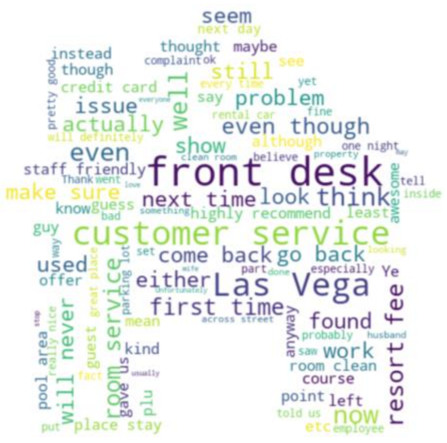
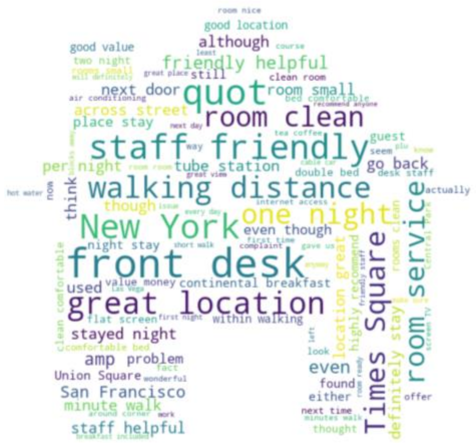
We performed two different analyses to answer our research question. We first used the Vader lexicon to assign sentiment scores to each reviews. After examining the findings of our sentiment analysis, we proceeded to build a model to predict ratings from sentiment scores.

1. *Sentiment Analysis*

Before we applied the Vader lexicon to our document-term matrix, we modified the lexicon to include certain new words. These new words and scores are shown below:

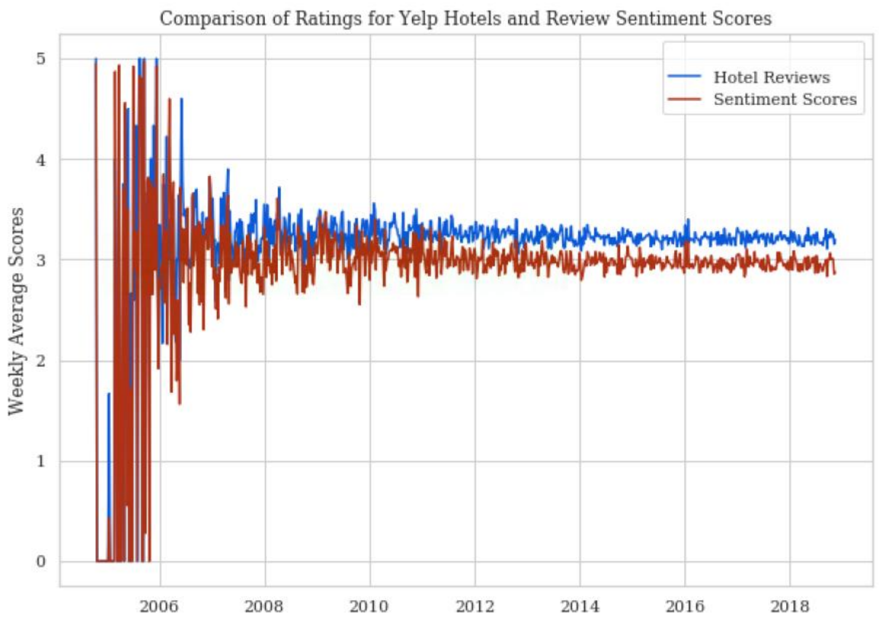
Words	**Scores**

against	-3
not	-3
nor	-3
no	-3
dirty	-4
messy	-4
mean	-1
unfortunately	-1
complaint	-4
complain	-4
clean	3
bad	-5
unclean	-3
friendly	4.5
issue	-2.5

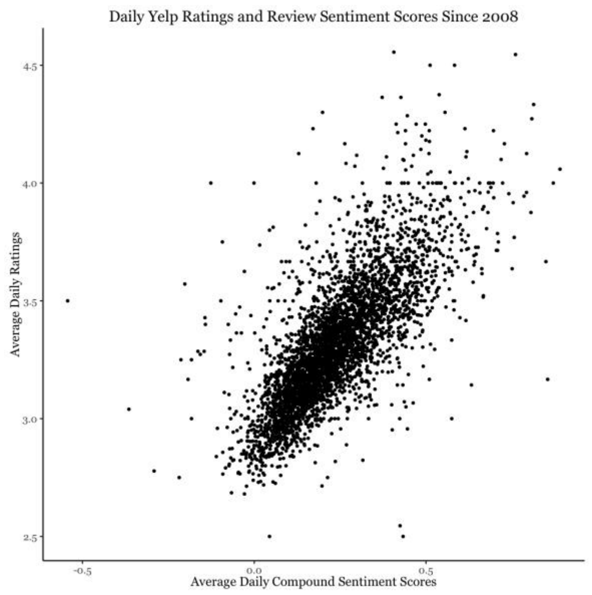
Word cloud displaying most common words in Yelp reviews Most common words in TripAdvisor reviews

All of the words in the table appeared these word clouds representing most frequent terms. They are not included in the original Vader lexicon, so we added them to make sure our sentiment scores can be more informative. After we applied the lexicon to the un-lemmatized document term matrix, we grouped the compound scores by week and by day. We then rescaled the compound score by multiplying the scores by 2.5 and adding 2.5. This allows the compound score to be scaled on the same range as the actual review ratings.



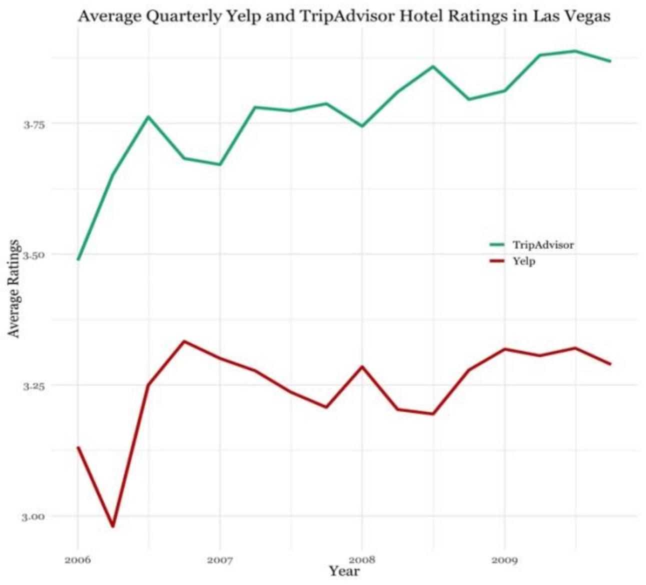
The averaged hotel star ratings and sentiment scores follow the same pattern, though the actual review ratings tend to be a little higher than the sentiment scores. However, the sentiment score is a little bit below the actual ratings. This pattern indicates the Yelp users are generally more forgiving, since they tend to give higher ratings than how they actually feel. In general, though, the ratings on Yelp truthfully represent how the users feel about the hotels they visited.

The scatterplot on the next page demonstrates the strength of the relationship between ratings and sentiments. The relationship appears tightly linear, and it resulted in an r^2 of approximately 0.55. Despite this, we instead elected to use a gradient boosted tree for reasons described in the next section.



2. *Predictive Analytics*

Our goal is to demonstrate scalability. That is, we want apply our boosted tree model to TripAdvisor data, which does not include review ratings. While the Yelp and TripAdvisor data sets cover varying years and geographic locations, they both contain hotel review data from 2006-2009 in Las Vegas. The visualization below shows the similarity in structure and mean trend for these time series, Interestingly, Yelp's average hotel ratings are far lower than that of TripAdvisor, both overall and for the city of Las Vegas, though the long-term pattern between these two time series is the same.



To test this similarity, we performed an Engle-Granger Cointegration test on the Yelp and TripAdvisor hotel ratings over time. This test yielded a p-value < 0.05, which implies we have sufficient evidence to conclude that the time series are indeed cointegrated. We suppose that the cointegration found in these time series holds for Yelp and TripAdvisor hotel ratings in general. Under this assumption, we applied the Yelp training data set to TripAdvisor as the testing set. Refer to the R-code in the appendix for details.

Why can we apply our model to another dataset? If we use the document-term matrix as our predictors, then it is very likely that words appearing in the Yelp dataset do not always appear in the TripAdvisor dataset, meaning these two data sets might have different predictors. This issue can be solved by "condensing" our document term matrix into a smaller matrix containing the most amount of information in the original dataset. Because word2vec can be an expensive and time-consuming approach, we will instead use singular-value decomposition to reduce our document term matrix into a small matrix containing 40 columns from what was originally hundreds of thousands. These 40 singular values contains 10% of the information stored in the original DTM[1]. This means that a practitioner may increase this number to yield even more accurate results.

We named the 40 singular values S1-S40 accordingly and applied this same process to TripAdvisor, allowing us to apply our model trained from the Yelp dataset to the TripAdvisor dataset directly. Refer to Section 3.7 of our Python code for more details.

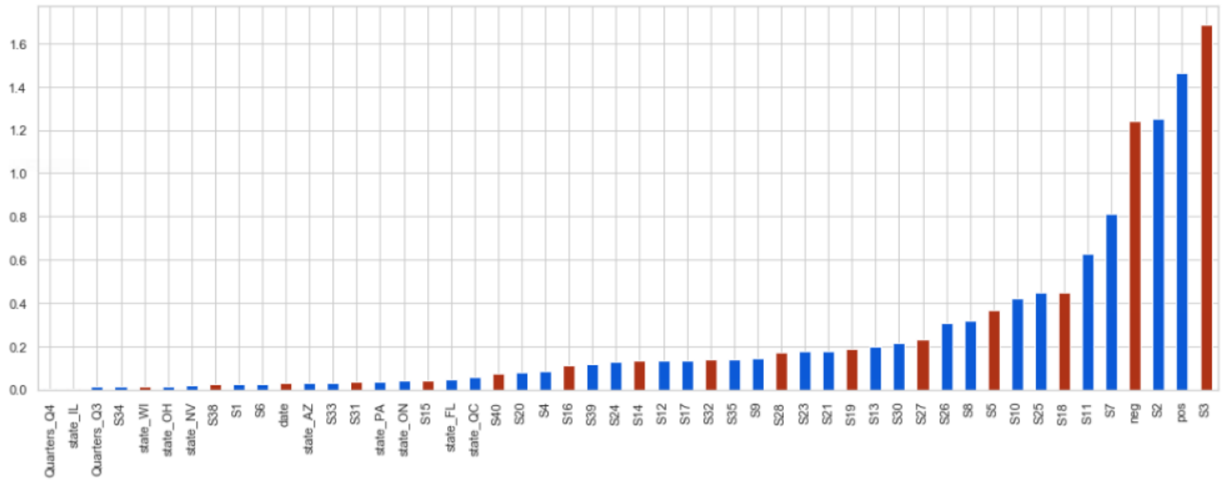
We did not predict review ratings solely from these singular values. Although that might give us a model with a high r^2 or low RMSE, but there is more information in the dataset that we can incorporate into our model. We also considered date (when the review was posted), state (where the hotel was located), as well as positive and negative sentiment scores as additional predictors. We converted each potential feature into a numeric variable in order to accomodate the boosted tree. For example, the date 3/1/2017 was represented as 2017.1643836 (2017 + (60/365)) in our training dataset. Similarly, state was transformed into a binary dummy variable to be included in the model. Additionally, using compound scores will not help us to predict rating, since one single review can contain positive and negative sentiments. If we separate these scores results, we can account for a review with a high positive sentiment that may not necessarily indicate the review does not contain something negative. Therefore, for the purpose of predictive analytics, we used positive and negative sentiment scores to predict hotel ratings.

After we added all 61 predictors, we recognized that is likely too many. Some variables appear to contribute nothing to predict hotel review ratings. To find the most important features, we initially built an elastic-net model, which is a hybrid of ridge and LASSO regression. This approach balances these feature-selection tools and accounts for potential correlation among variables.

Before building a model, we partitioned our data into training (70%) and testing (30%) sets. After tuning the elastic-net model using 10-fold cross validation, we came up with a model with $\alpha = 0.03$ and $\lambda = 0.01$. The r^2 for the holdout set was around 0.54, with a validation RMSE

[1] In other words, the sum of squares of the singular values is equal to the total variance in the original DTM.

of approximately 1.05. This indicates our elastic-net model only explains 54% of the total information in the data set. While it is also not a good prediction model since it contains relatively high prediction error, it demonstrates important features. By observing the absolute value of the model coefficients, we can determine which variables are useful, as shown below.



Most of the singular values created from the decomposed DTM are significant, as well as “negative” and “positive” sentiment scores. However, time and geography appear not to contribute significantly. We decided to remove 11 variables having no feature importance. Although some variables shown in the plot above have very tiny coefficients, we still kept them because they contribute some predictive power. We will use a more powerful algorithm to build our final model and we will use these 50 variables as our main predictors.

Because the hotel review ratings should be between 0 and 5, we do not want our model to give us a prediction greater than 5 or below 0. Therefore, we cannot use a linear regression model because the linear regression model will extrapolates (i.e. it will give us predictions higher than 5). On the contrary, predictions from tree models are restricted to the range of the target variable used for training. All the predictions generated from tree models will be between 0 and 5. This is the reason why we decided to use a gradient boosted tree model as our final prediction model.

The algorithm we used is called XGBoost (Extreme Gradient Boosting), which builds a tree using some randomly-generated parameters. Then, it calculates the residuals (prediction errors) of the first tree and uses them as the target variable for the next tree it builds. The predicted values are calculated by combining the predictions of all trees, but the predictions of previous trees are always given a higher weight. XGBoost has eight hyper tuning parameters, including a learning rate parameter, maximum depth and minimum child weight parameters, and regularizations, combining aspects from ridge, random forests, and neural networks. We performed 10-fold cross validation twice to tune the parameters. Refer to Section 6 of our Python code for more information about parameter tuning. The best parameters are shown below:

Parameter	**Value**
learning rate	0.05
n estimators	1750
max depth	7
Min child weight	5
gamma	0.05
subsample	0.8
colsample bytree	0.8
reg alpha	30

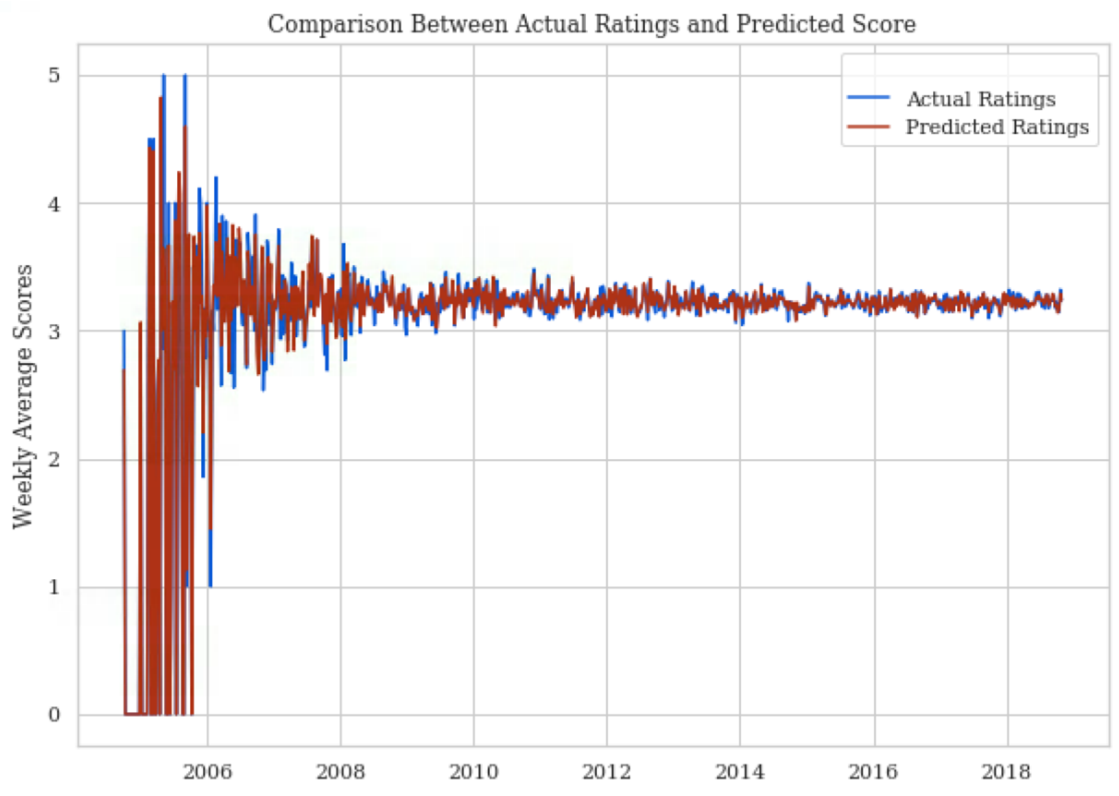
Based on our model, we calculated the RMSE and r^2 statistics shown below:

	Training Set	**Validation Set**
R-Squared	0.7985	0.7025
RMSE	0.6986	0.8487

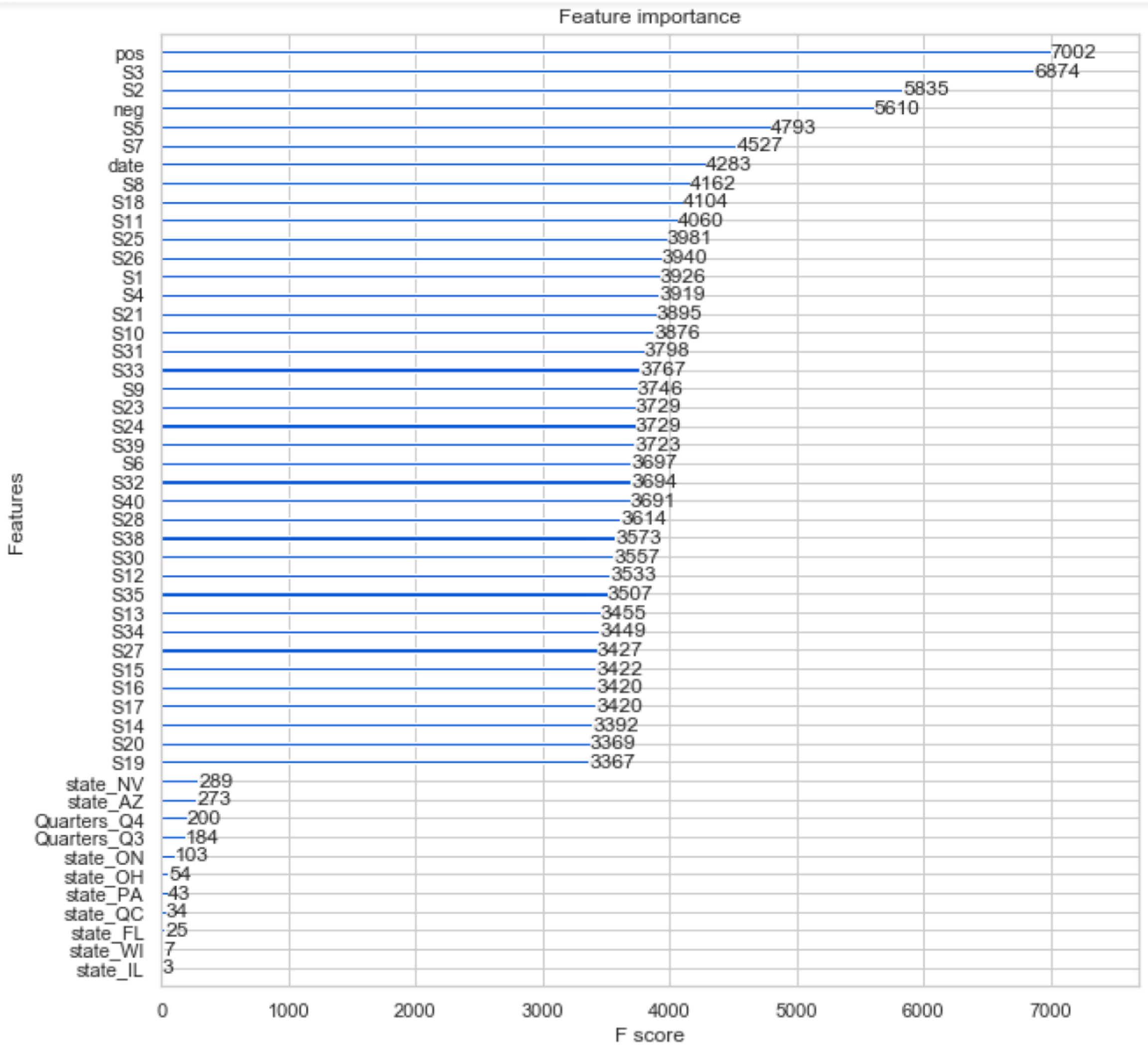
We noticed that our target variable, review ratings, can only be integers (i.e. users cannot give a hotel a three and a half star). Based on this finding, we rounded our predicted values to integers and recalculated the RMSE and r^2, to obtain the following:

	Validation Set
R-Squared	0.7697
RMSE	0.7468

The statistics shown above indicates that our model can explain over 70% of the total variability of Yelp reviews. Indeed, our model generates relatively reliable predictions. This time series displays actual ratings plotted against predicted values.



We then examined the variable importance of our model. The variable importance plot is shown below. Note that how XGBoost calculates variable importance is different from the way importance is calculated in our previous elastic net model. In our previous elastic net example, the variable importance of one variable is simply the effect size of that predictor. However, since XGBoost is a tree based algorithm. It would be more reasonable to calculate feature importance by summing up how many times each feature is split on. From the plot below, we can clearly see the singular values created from the DTM are still very important to our prediction. The positive sentiment score is the most important variable, meaning that it contributed the most to predicting hotel review ratings. Apart from the positive score, the negative score and date are also pretty important predictors. On the contrary, all the state and quarter variables do not significantly contribute to predicting review ratings. This variable importance scree-plot confirms our previous finding from the sentiment analysis that the review ratings truthfully represent the users' emotion, whether it's positive or negative. The content of the reviews will contribute significantly to determining the actual ratings.



We them applied our model to the TripAdvisor dataset to see how our model can be used to predict review ratings for a new data source. **We ended up getting a RMSE of 0.69.** This shows that our model can predict hotel ratings with a pretty low prediction error.

Conclusion

1. *Pros and Cons*

Our analysis can be very helpful to hotel owners. As mentioned, Yelp and TripAdvisor can compare actual ratings to sentiment scores to identify customers that behave inconsistently. In addition, hotel owners will be able to assess how satisfied their customers really are (based on a scale of 5) by simply feeding in text reviews. Because our predicted ratings are continuous (as opposed to discrete), our ratings will be more granular in nature. Compared to traditional review ratings shown on Yelp and TripAdvisor's websites, our predicted ratings will allow customers to more clearly distinguish hotels with differences in quality.

However, our model is not perfect. When we performed singular value decomposition, we only used the first 40 singular values. These singular values in total only explains about 10% of the information in the original document term matrix. If we had a month available to run our model, we could then use the first 100 or 200 singular values instead. By doing so, our model could perform better. Moreover, instead of using singular value decomposition, we could use word2vector to perform dimension reduction. If we had the time, word2vector could have reduced the number of features more accurately.

2. *Deployment*

Because our end result is a model, we can it to Yelp or TripAdvisor in the form of a web application. Users of the application will be able to input text data into the application, and our model would return the predicted result. Yelp and TripAdvisor can implement such applications into their Enterprise Resource Management system. They could give individual hotel owners access to the application so that they can monitor their customer experiences over time. Of course, access to the model should not be free. Yelp and TripAdvisor can learn from companies like NASDAQ, who sell their forecasts to their customers. They could also publish the predicted ratings on their website to help customers locate the best hotels. They could even brand our predicted ratings as “Y-estimate” or “T-estimate” (i.e. Zillow has a Z-estimate score, which is the predicted market value for an individual home).

3. *Possible financial gains & Decision making improvements*

There are plenty of risk factors that Yelp mentioned in their SEC form which they used to make investing decisions on their official website. One of the risk factors related to what we analyze in this report is “*the quantity and quality of the content contributed by our users, as well as the perceived distribution of such content across the categories of businesses on our platform*” (Sec Form). Our model can help Yelp users obtain more accurate online review rates, which means it could help Yelp perform better in the online review market. The reputation of Yelp will improve, catalyzing more traffic to the website. Since Yelp derives a substantial majority of their revenue based on the user engagement with displayed ads, the increased web traffic will result in financial gain.

To estimate the cost of this model, we found their capitalized website and software development cost is around 20 million dollars in the year of 2018. This means Yelp pays a lot of attention towards improving their desktop and mobile app services. Since the brand image and reputation are of high importance to Yelp, stakeholders can be persuaded to implement our model due to the utility and insight it cultivates, ultimately resulting in improved user traffic and revenue.

APPENDEX:

Works Cited

Horrigan J. *Pew Internet and American Life Project Report*. Pew Research Center; 2008. Online shopping.
Hafsa J. *Stemming and Lemmatization in Python.* DataCamp; 2018. Community Tutorials.
Sec Filing Details of Yelp, Form 10-Q Table of Contents, *ITEM 1A. RISK FACTORS, 2019*

R Code for Visualizations, Time Series Analysis, Cointegration Test

NOTE: All data cleaning and analytics completed in Python are available in separately-attached .ipnyb files.

```
# Loading relevant packages
library(tidyverse)
library(lubridate)
library(zoo)
library(aTSA)
library(forecast)
library(gridExtra)
library(extrafont)
loadfonts()

# Setting working directory
setwd("~/Desktop/ISA 414 - Big Data/Project")

# Importing data
yelp <- read.csv("YelpHotelRev.csv")
tripAdvisor <- read.csv("TripAdvisorRev.csv")

# Constructing new column which converts dates to years
yelp$quarter <- as.yearqtr(yelp$date)
tripAdvisor$quarter <-
     as.yearqtr(parse_date_time(tripAdvisor$Date, orders =
"mdy"))

# Filtering Yelp for only 2006-2009
yelp <- yelp[is.na(yelp$quarter) == FALSE, ]
yelp_filtered <- yelp[str_detect(yelp$quarter, "2006") |
                      str_detect(yelp$quarter, "2007") |
                      str_detect(yelp$quarter, "2008") |
                      str_detect(yelp$quarter, "2009"), ]

# Filtering TripAdvisor for only 2006-2009
tripAdvisor <- tripAdvisor[is.na(yelp$quarter) == FALSE, ]
tripAdvisor_filtered <-
tripAdvisor[str_detect(tripAdvisor$quarter, "2006") |
            str_detect(tripAdvisor$quarter, "2007") |
            str_detect(tripAdvisor$quarter, "2008") |
            str_detect(tripAdvisor$quarter, "2009"),]
```

```
# Grouping by quarter and calculating average rating for Yelp
and TripAdvisor
yelp_summary <- yelp_filtered %>% select(stars_business,
     stars_reviews, quarter) %>%
     group_by(quarter) %>%
     summarise(avg_business_yelp = mean(stars_business))

tripAdvisor_summary <- tripAdvisor_filtered %>%
     select(overall_ratingsource, quarter) %>%
     group_by(quarter) %>%
     summarise(avg_reviews_ta = mean(overall_ratingsource))
tripAdvisor_summary <- tripAdvisor_summary[1:16, ]

# Merging data
merged_summary <- merge(yelp_summary, tripAdvisor_summary, by =
     "quarter")

# Filtering for Las Vegas
yelp_filtered <- yelp %>%
  filter(yelp$city == "Las Vegas")
tripAdvisor_filtered <- tripAdvisor %>%
  filter(tripAdvisor$city == "las vegas")

# Grouping by quarter and calculating average ratings
yelp_summary <- yelp_filtered %>% select(stars_business,
     stars_reviews, quarter) %>%
     group_by(quarter) %>%
     summarise(avg_business_yelp = mean(stars_business))

tripAdvisor_summary <- tripAdvisor_filtered %>%
     select(overall_ratingsource, quarter) %>%
     group_by(quarter) %>%
     summarise(avg_reviews_ta = mean(overall_ratingsource))

# Merging data
LV_merged_summary <- merge(yelp_summary, tripAdvisor_summary, by
     = "quarter")
LV_merged_summary <- LV_merged_summary[4:19,]

# Exporting visualization upon knit
```

```
tiff("Las Vegas Time Series.tiff", width = 7, height = 7,
     pointsize = 1/500, units = 'in', res = 500)

# Constructing Las Vegas time series, juxtaposed with original
ggplot(LV_merged_summary, aes(as.Date(quarter))) +
     geom_line(aes(y = LV_merged_summary$avg_reviews_ta, color =
     "#00AF87"), size = 1.2) + geom_line(aes(y =
     LV_merged_summary$avg_business_yelp, color = "#c41200"),
     size = 1.2) +
     ggtitle("Average Quarterly Yelp and TripAdvisor Hotel
     Ratings in Las Vegas") +
     xlab("Year") + ylab("Average Ratings") + theme_minimal() +
     theme(legend.position="top") + scale_color_identity(name =
     element_blank(), labels = c("TripAdvisor", "Yelp"), guide =
     "legend") + theme(plot.title = element_text(hjust = 0.5,
     size = 14), text = element_text(family = "Georgia"),
     legend.text = element_text(size = 9), legend.position =
     c(0.818, 0.575), legend.key.size = unit(0.5, 'cm'),
     axis.title = element_text(size = 12))

# Converting quarterly data to time series
y <- ts(LV_merged_summary$avg_business_yelp, frequency = 4)
x <- ts(LV_merged_summary$avg_reviews_ta, frequency = 4)

# Assesing number of differences for each time series (outputs
1)
ndiffs(y)
ndiffs(x)

# Performing cointegration test
coint.test(y, x, d = 1)

# Standardizing time series
x <- y + mean(x - y)
LV_merged_summary$avg_reviews_ta <-
     LV_merged_summary$avg_business_yelp +
     mean(LV_merged_summary$avg_reviews_ta -
     LV_merged_summary$avg_business_yelp)

# Re-naming variables for cleaner Yelp data set
```

```
yelp_renamed <- yelp %>% select(Name = name, City = city, State
=
     state, ReviewCount = review_count,
     Date = date, Review = text, Rating = stars_reviews)

yelp_renamed$Date <-
ymd(as.Date(as.character(yelp_renamed$Date)))

# Exports cleaner data set
yelpFinal <- write.csv(yelp_renamed)
```

(At this point, we continued to manipulate the data set in Python Code. Below is the scatterplot visualization from the final cleaned data set)

```
# Loading final Yelp Data set
yelp <- read.csv("FinalDatasetYelp.csv")

# Filtering reviews past 2008
yelpFinal <- yelp[yelp$date >= 2008,]
yelpFinal <- yelpFinal %>%
  group_by(date) %>%
  summarise(meanCompound = mean(compound), meanRating =
mean(stars_reviews))
# Ordering data set by date
yelpFinal <- yelpFinal[order(yelpFinal$date), ]

# Exporting visualization upon knit
tiff("Scatterplot.tiff", width = 7, height = 7, pointsize =
1/300, units = 'in', res = 300)

# Constructing scatterplot
ggplot(data = yelpFinal, aes(x = meanCompound, y = meanRating))
+
     geom_point(size = 0.8) + theme_classic() + theme(plot.title
     = element_text(hjust = 0.5), text = element_text(family =
     "Georgia")) + ggtitle("Daily Yelp Ratings and Review
     Polarities Since 2008") + labs(x = "Average Daily Compound
     Polarity Score", y = "Average Daily Rating")
```