USE CASE STUDY REPORT

Group No.: Group 11

Student Names: Manuel Curiel and Shriram Karthikeyan

Executive Summary:

The goal of this case study is to classify a given restaurant's user rating so that we can use the predicted rating to suggest restaurants to users. The dataset used for this case study was provided by Yelp as a part of the Yelp Dataset Challenge, the total size of the data amounts to 10 GB. To make our case study computationally feasible, we have limited our problem to the scope of only restaurants, and we reduced the amount of reviews to 10% (278,716 of 2,787,169). The data contained 14 continuous variables and 18 categorical variables (the categorical variables were converted to dummies), PCA was applied on continuous variables. We used the following data mining techniques to solve the problem: random forests, classification trees, k-nearest neighbors, naïve Bayes, and Linear Discriminant Analysis. Confusion matrices and gain charts were used to measure performance and random forest provided the best results taking sensitivity into consideration. The original problem was converted to a two-class classification to segregate the restaurants into good and bad restaurants. To improve our model, we would recommend taking the sentiment of the text of the review into consideration and the complete dataset which will improve the classification.

I. Background and Introduction

Problem

Looking for reviews before going out to dinner has become a routine for most of us. Searching for photos and comments on Google or Yelp are crucial to make an informed decision on where to go. And if we do decide to try out the new restaurant/bar, we definitely want to know what others think about the food and drinks we would consume. This review mining activity can be really frustrating, requiring users to go through dozens of reviews to try and understand if a particular place would match their preferences. However, we believe that the reviews we give restaurants in Yelp already have a great amount of information about our likes and dislikes, and a data mining technique could be used to help us find a place that matches our preferences.

Over the years Yelp has become a widely popular choice for people to read and write reviews about their experiences. They have recently made more than 7 GB of data about reviews, businesses and users available for students and researchers to work. We believe this data could be used by Yelp to predict the ranking a user will give to a particular restaurant/bar.

The goal of your study

The objective of this case study is to create a model that would predict the number of stars a user will give to a give food business. And use this information to make accurate food services recommendations to users.

The possible solution

The starting point solution for this case study is to create a five class classification model that would predict the ranking a user will give to a business. The training of the data will be supervised, and will consist of several data mining techniques:

- Decision trees
- Random forests
- k nearest neighbors
- Naïve Bayes
- Linear Discriminant Analysis
- Logit classification

The performance of each technique will be evaluated using the following methods:

- Confusion matrix
    - Accuracy values
    - Sensitivity
- Gain chart
- Lift chart

II. Data Exploration and Visualization

Merging the data sets

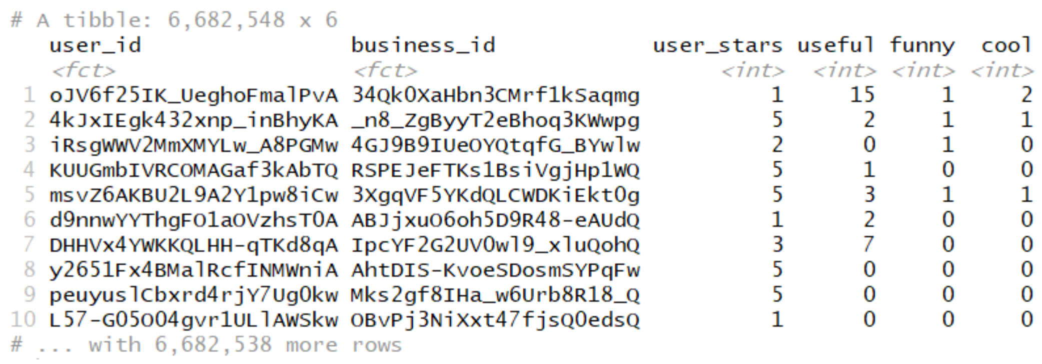
Provide brief description of techniques used to explore the data including: basic charts, distribution plots, correlations, missing values, rescaling, aggregation, hierarchies, zooming, filtering, etc.

The following files were downloaded from Yelps website:

- Review.json
- Business.json
- Users.json

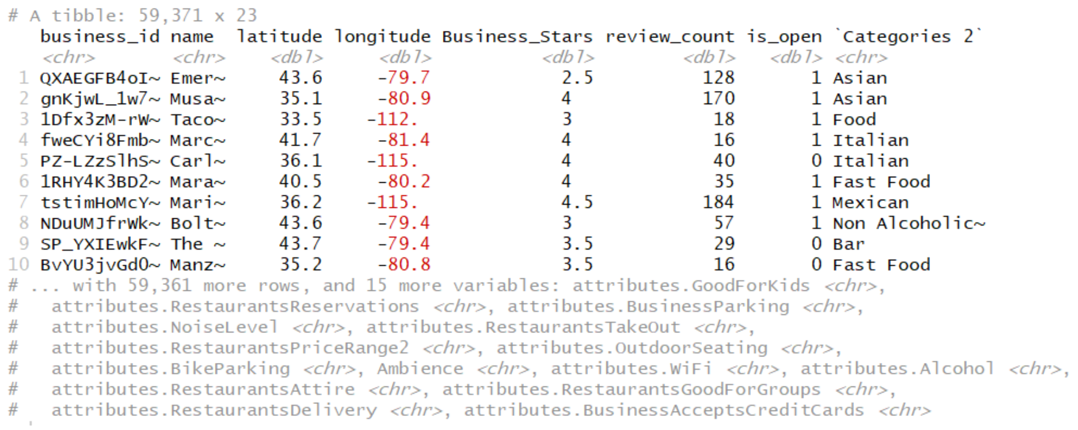
These three files had a total size of 7 Gb, which made it very hard for R to process. The reviews data was especially heavy with more than 6 million reviews. Our first challenge was to "merge" all this information in a new data frame, and then preprocess the data to apply the data mining techniques. The following images are a small sample of this data sets, as they were received from Yelp:

Review.json:



```
# A tibble: 6,682,548 x 6
   user_id                business_id            user_stars useful funny  cool
   <fct>                  <fct>                       <int>  <int> <int> <int>
 1 oJV6f25IK_UeghoFmalPvA 34Qk0XaHbn3CMrf1kSaqmg          1     15     1     2
 2 4kJxIEgk432xnp_inBhyKA _n8_ZgByyT2eBhoq3KWwpg          5      2     1     1
 3 iRsgWWV2MmXMYLw_A8PGMw 4GJ9B9IUeOYQtqfG_BYwlw          2      0     1     0
 4 KUUGmbIVRCOMAGaf3kAbTQ RSPEJeFTKs1BsiVgjHp1WQ          5      1     0     0
 5 msvZ6AKBU2L9A2Y1pw8iCw 3XgqVF5YKdQLCWDKiEkt0g          5      3     1     1
 6 d9nnwYYThgFO1aOVzhsT0A ABJjxuO6oh5D9R48-eAUdQ          1      2     0     0
 7 DHHVx4YWKKQLHH-qTKd8qA IpcYF2G2UV0wl9_xluQohQ          3      7     0     0
 8 y2651Fx4BMalRcfINMWniA AhtDIS-KvoeSDosmSYPqFw          5      0     0     0
 9 peuyuslCbxrd4rjY7Ug0kw Mks2gf8IHa_w6Urb8R18_Q          5      0     0     0
10 L57-G05O04gvr1ULlAWSkw OBvPj3NiXxt47fjsQ0edsQ          1      0     0     0
# ... with 6,682,538 more rows
```

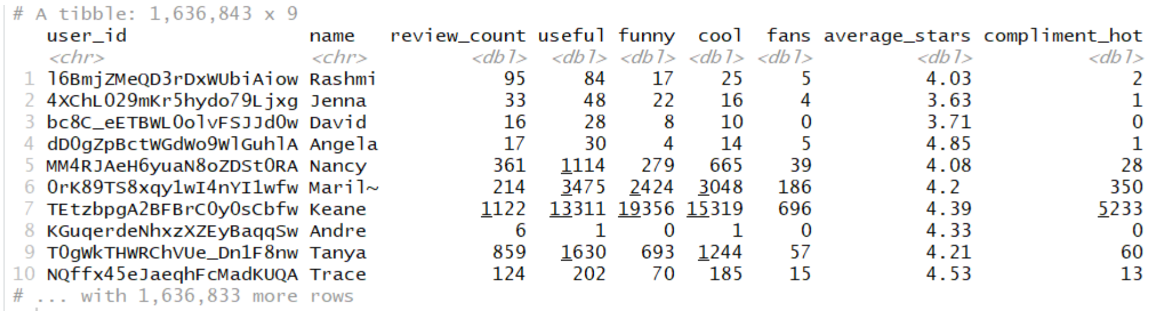
Business.json



```
# A tibble: 59,371 x 23
   business_id name  latitude longitude Business_Stars review_count is_open `Categories 2`
   <chr>       <chr>    <dbl>     <dbl>          <dbl>        <dbl>   <dbl> <chr>
 1 QXAEGFB4oI~ Emer~     43.6     -79.7          2.5            128       1 Asian
 2 gnKjwL_1w7~ Musa~     35.1     -80.9          4              170       1 Asian
 3 1Dfx3zM-rW~ Taco~     33.5    -112.           3               18       1 Food
 4 fweCYi8Fmb~ Marc~     41.7     -81.4          4               16       1 Italian
 5 PZ-LZzSlhS~ Carl~     36.1    -115.           4               40       0 Italian
 6 1RHY4K3BD2~ Mara~     40.5     -80.2          4               35       1 Fast Food
 7 tstimHoMcY~ Mari~     36.2    -115.           4.5            184       1 Mexican
 8 NDuUMJfrWk~ Bolt~     43.6     -79.4          3               57       1 Non Alcoholic~
 9 SP_YXIEwkF~ The ~     43.7     -79.4          3.5             29       0 Bar
10 BvYU3jvGd0~ Manz~     35.2     -80.8          3.5             16       0 Fast Food
# ... with 59,361 more rows, and 15 more variables: attributes.GoodForKids <chr>,
#   attributes.RestaurantsReservations <chr>, attributes.BusinessParking <chr>,
#   attributes.NoiseLevel <chr>, attributes.RestaurantsTakeOut <chr>,
#   attributes.RestaurantsPriceRange2 <chr>, attributes.OutdoorSeating <chr>,
#   attributes.BikeParking <chr>, Ambience <chr>, attributes.WiFi <chr>, attributes.Alcohol <chr>,
#   attributes.RestaurantsAttire <chr>, attributes.RestaurantsGoodForGroups <chr>,
#   attributes.RestaurantsDelivery <chr>, attributes.BusinessAcceptsCreditCards <chr>
```

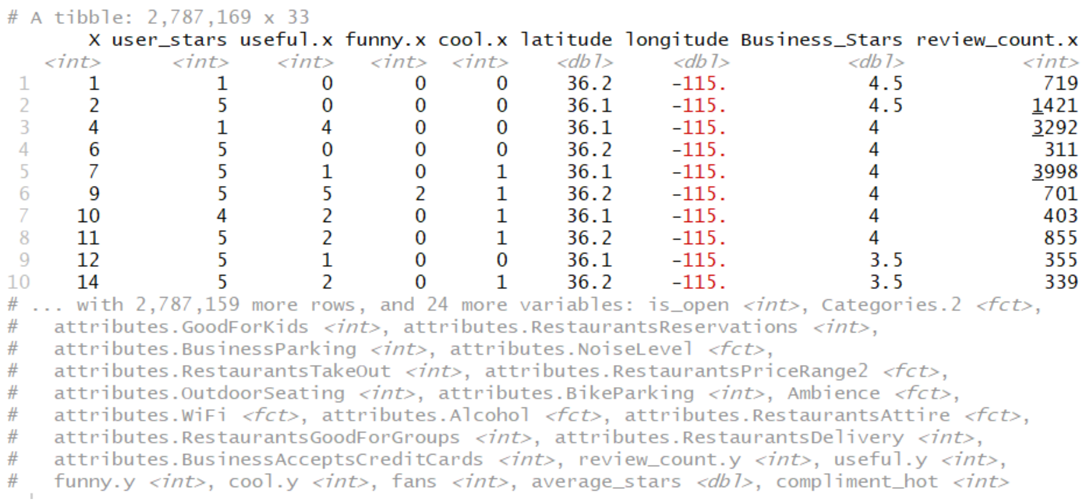
This data set required a lot of cleaning for NA values, and formatting of business attributes values. It originally had around 60 different variables, but most of them had NA values, and were removed from the data set. Furthermore, the original data had more than 100,000 business that were not in the food service business, and were removed from the analysis. The "Categories 2" variable was reduced from 512 factors to just 19 to make the dummy variable generation feasible.

Users.json



```
# A tibble: 1,636,843 x 9
   user_id                name   review_count useful funny  cool  fans average_stars compliment_hot
   <chr>                  <chr>         <dbl>  <dbl> <dbl> <dbl> <dbl>         <dbl>          <dbl>
 1 l6BmjZMeQD3rDxWUbiAiow Rashmi           95     84    17    25     5          4.03              2
 2 4XChL029mKr5hydo79Ljxg Jenna            33     48    22    16     4          3.63              1
 3 bc8C_eETBWL0olvFSJJd0w David            16     28     8    10     0          3.71              0
 4 dD0gZpBctWGdWo9WlGuhlA Angela           17     30     4    14     5          4.85              1
 5 MM4RJAeH6yuaN8oZDSt0RA Nancy           361   1114   279   665    39          4.08             28
 6 0rK89TS8xqy1wI4nYI1wfw Maril~          214   3475  2424  3048   186          4.2             350
 7 TEtzbpgA2BFBrC0y0sCbfw Keane          1122  13311 19356 15319   696          4.39           5233
 8 KGuqerdeNhxzXZEyBaqqSw Andre             6      1     0     1     0          4.33              0
 9 T0gWkTHWRChVUe_Dn1F8nw Tanya           859   1630   693  1244    57          4.21             60
10 NQffx45eJaeqhFcMadKUQA Trace           124    202    70   185    15          4.53             13
# ... with 1,636,833 more rows
```

This three data sets were merged into the file "Yelp_Dataset.csv" using the "merge" function in R. After the merge, the size of the data set was 6,682,548 x 33. However, several predictors had NA values that had to be solved. Since the number of cases with no NA values is almost 3 million, we decided to just eliminate cases that had predictors with NA values. A sample of the final result of this data processing can be seen in the following image:



```
# A tibble: 2,787,169 x 33
       X user_stars useful.x funny.x cool.x latitude longitude Business_Stars review_count.x
   <int>      <int>    <int>   <int>  <int>    <dbl>     <dbl>          <dbl>          <int>
 1     1          1        0       0      0     36.2     -115.            4.5            719
 2     2          5        0       0      0     36.1     -115.            4.5           1421
 3     4          1        4       0      0     36.1     -115.            4             3292
 4     6          5        0       0      0     36.2     -115.            4              311
 5     7          5        1       0      1     36.1     -115.            4             3998
 6     9          5        5       2      1     36.2     -115.            4              701
 7    10          4        2       0      1     36.1     -115.            4              403
 8    11          5        2       0      1     36.2     -115.            4              855
 9    12          5        1       0      0     36.1     -115.            3.5            355
10    14          5        2       0      1     36.2     -115.            3.5            339
# ... with 2,787,159 more rows, and 24 more variables: is_open <int>, Categories.2 <fct>,
#   attributes.GoodForKids <int>, attributes.RestaurantsReservations <int>,
#   attributes.BusinessParking <int>, attributes.NoiseLevel <fct>,
#   attributes.RestaurantsTakeOut <int>, attributes.RestaurantsPriceRange2 <fct>,
#   attributes.OutdoorSeating <int>, attributes.BikeParking <int>, Ambience <fct>,
#   attributes.WiFi <fct>, attributes.Alcohol <fct>, attributes.RestaurantsAttire <fct>,
#   attributes.RestaurantsGoodForGroups <int>, attributes.RestaurantsDelivery <int>,
#   attributes.BusinessAcceptsCreditCards <int>, review_count.y <int>, useful.y <int>,
#   funny.y <int>, cool.y <int>, fans <int>, average_stars <dbl>, compliment_hot <int>
```

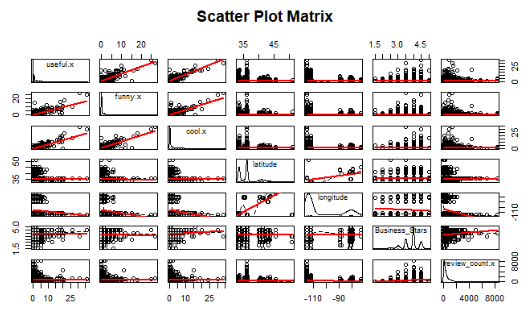
III. Data Preparation and Preprocessing

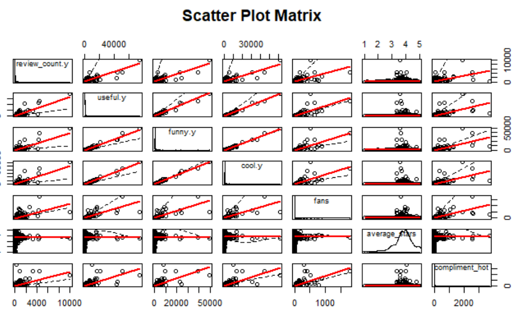
Categorical variables

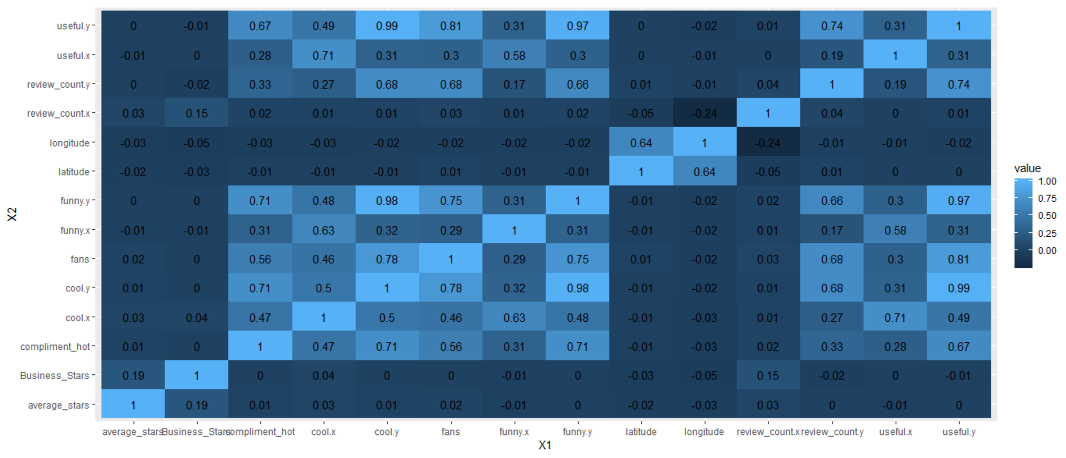
The data has 14 continuous variables and 18 categorical variables (without considering index column "X"). The categorical variables will have to be transformed into dummies using the fastDummies package in R, after this the data increased from 33 to 64 variables.

Continuous variables

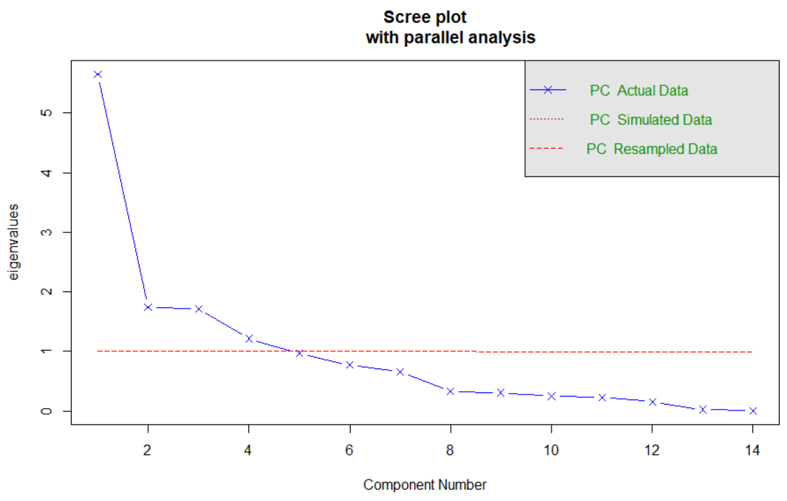
We performed PCA with the intention of reducing the size of the data set. This will probably reduce the data processing volume for the algorithms, and solve multicollinearity issues that may be present in the data set. In order to asses if our assumptions were correct, we did a scatterplot and correlation heat map of this variables:

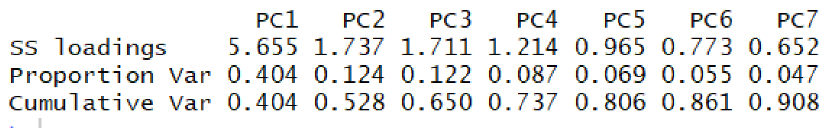






This shows high levels of correlation mainly between some of the variables. PCA will be applied to address this issue. The following image shows the screeplot used to decide how many PC to use:

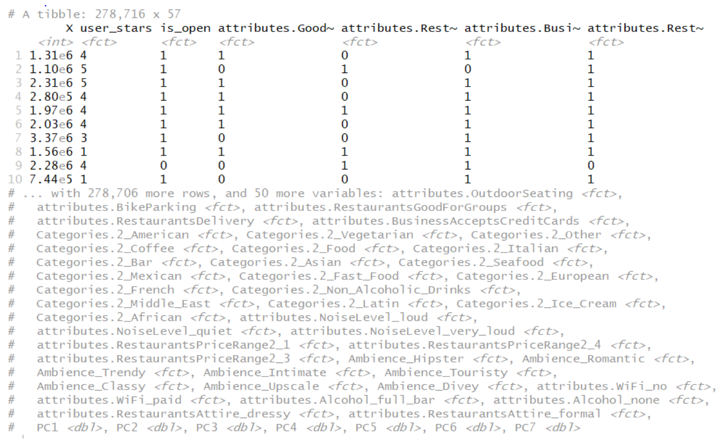




```
                PC1   PC2   PC3   PC4   PC5   PC6   PC7
SS loadings    5.655 1.737 1.711 1.214 0.965 0.773 0.652
Proportion Var 0.404 0.124 0.122 0.087 0.069 0.055 0.047
Cumulative Var 0.404 0.528 0.650 0.737 0.806 0.861 0.908
```

The screeplot recommended taking only 4 factors, however, this just represents roughly 74% of the total variance. So we decided that we were going to use 7 Principal Components representing 91% of the variance. We later used the PCA scores, and merged them into our original data set, reducing its size to 2,787,169 x 57.

Lastly, before separating our data into training, validation and testing, we reduced the amount of cases to 10% (278,716 of 2,787,169) in order to reduce data processing time. We took random samples of 50% training, 30% validation and 20% testing for our models. The following image shows a summary of the data after all the pre-processing tasks:

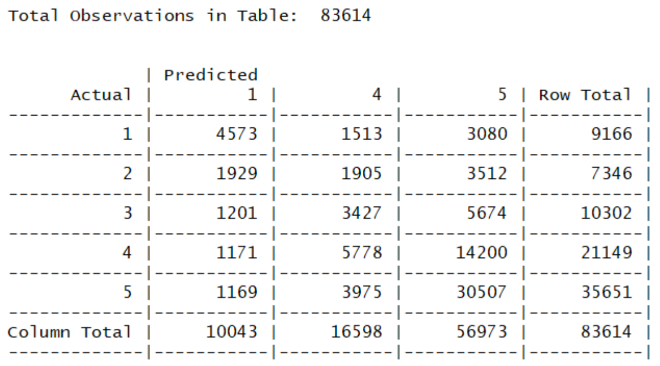


```
# A tibble: 278,716 x 57
         X user_stars is_open attributes.Good~ attributes.Rest~ attributes.Busi~ attributes.Rest~
     <int> <fct>      <fct>   <fct>            <fct>            <fct>            <fct>
 1 1.31e6 4          1       1                0                1                1
 2 1.10e6 5          1       0                1                0                1
 3 2.31e6 5          1       1                0                1                1
 4 2.80e5 4          1       1                0                1                1
 5 1.97e6 4          1       1                1                1                1
 6 2.03e6 4          1       1                0                1                1
 7 3.37e6 3          1       0                0                1                1
 8 1.56e6 1          1       1                1                1                1
 9 2.28e6 4          0       0                1                1                0
10 7.44e5 1          1       0                0                1                1
# ... with 278,706 more rows, and 50 more variables: attributes.OutdoorSeating <fct>,
#   attributes.BikeParking <fct>, attributes.RestaurantsGoodForGroups <fct>,
#   attributes.RestaurantsDelivery <fct>, attributes.BusinessAcceptsCreditCards <fct>,
#   Categories.2_American <fct>, Categories.2_Vegetarian <fct>, Categories.2_Other <fct>,
#   Categories.2_Coffee <fct>, Categories.2_Food <fct>, Categories.2_Italian <fct>,
#   Categories.2_Bar <fct>, Categories.2_Asian <fct>, Categories.2_Seafood <fct>,
#   Categories.2_Mexican <fct>, Categories.2_Fast_Food <fct>, Categories.2_European <fct>,
#   Categories.2_French <fct>, Categories.2_Non_Alcoholic_Drinks <fct>,
#   Categories.2_Middle_East <fct>, Categories.2_Latin <fct>, Categories.2_Ice_Cream <fct>,
#   Categories.2_African <fct>, attributes.NoiseLevel_loud <fct>,
#   attributes.NoiseLevel_quiet <fct>, attributes.NoiseLevel_very_loud <fct>,
#   attributes.RestaurantsPriceRange2_1 <fct>, attributes.RestaurantsPriceRange2_4 <fct>,
#   attributes.RestaurantsPriceRange2_3 <fct>, Ambience_Hipster <fct>, Ambience_Romantic <fct>,
#   Ambience_Trendy <fct>, Ambience_Intimate <fct>, Ambience_Touristy <fct>,
#   Ambience_Classy <fct>, Ambience_Upscale <fct>, Ambience_Divey <fct>, attributes.WiFi_no <fct>,
#   attributes.WiFi_paid <fct>, attributes.Alcohol_full_bar <fct>, attributes.Alcohol_none <fct>,
#   attributes.RestaurantsAttire_dressy <fct>, attributes.RestaurantsAttire_formal <fct>,
#   PC1 <dbl>, PC2 <dbl>, PC3 <dbl>, PC4 <dbl>, PC5 <dbl>, PC6 <dbl>, PC7 <dbl>
```

IV. Data Mining Techniques and Implementation

As stated in the problem definition, the first step was to try different supervised learning classification techniques, for this 5 class classification problem. All the confusion matrix and accuracy results correspond to the validation data. The result of this first attempt is shown next:

Classification tree

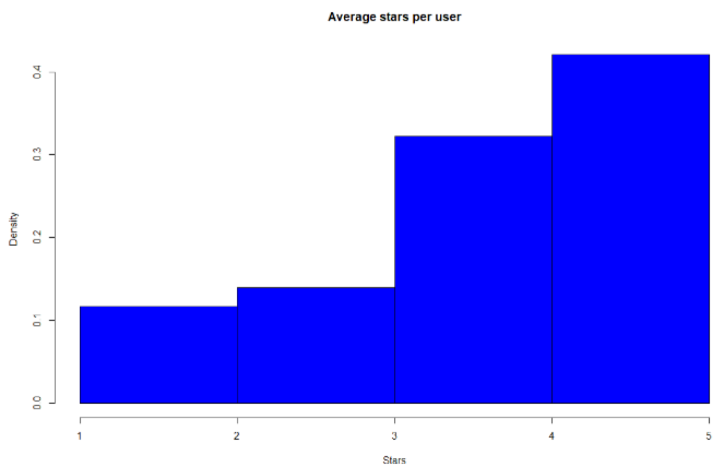


```
Total Observations in Table:  83614 

 
             | Predicted 
      Actual |         1 |         4 |         5 | Row Total | 
-------------|-----------|-----------|-----------|-----------|
           1 |      4573 |      1513 |      3080 |      9166 | 
-------------|-----------|-----------|-----------|-----------|
           2 |      1929 |      1905 |      3512 |      7346 | 
-------------|-----------|-----------|-----------|-----------|
           3 |      1201 |      3427 |      5674 |     10302 | 
-------------|-----------|-----------|-----------|-----------|
           4 |      1171 |      5778 |     14200 |     21149 | 
-------------|-----------|-----------|-----------|-----------|
           5 |      1169 |      3975 |     30507 |     35651 | 
-------------|-----------|-----------|-----------|-----------|
Column Total |     10043 |     16598 |     56973 |     83614 | 
-------------|-----------|-----------|-----------|-----------|
```

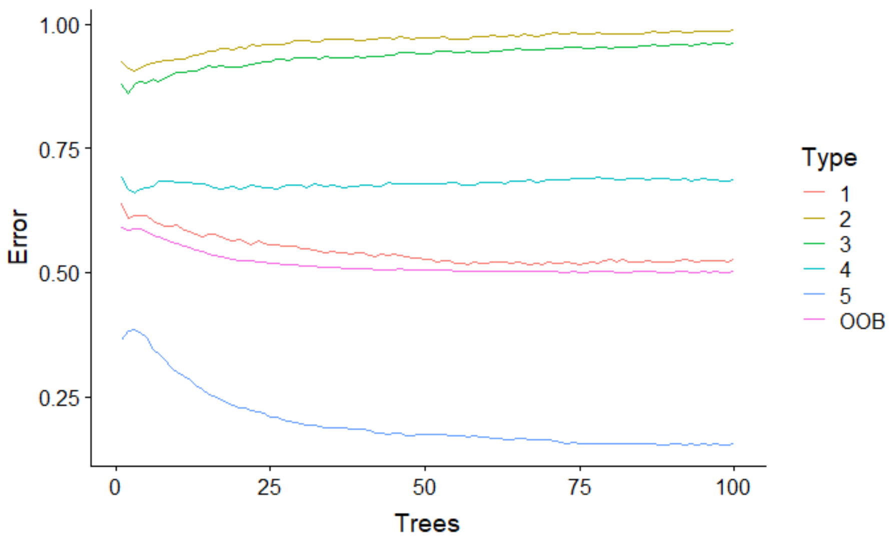
The accuracy result for this technique has a low value of 49%. Some interesting things to observe from the confusion matrix is that the model did not make any prediction of 2 and 3 ratings, and did a very good job accurately predicting 5 stars' ratings (86% sensitivity).

The reason for this behavior might be that most users tend to rate restaurants between 3 and 5 stars. There are not many values between 1 and 3 for the model to learn how to predict. The following histogram shows the frequency of average stars given by user, close to 80% of the users have an average between 3 and 5.



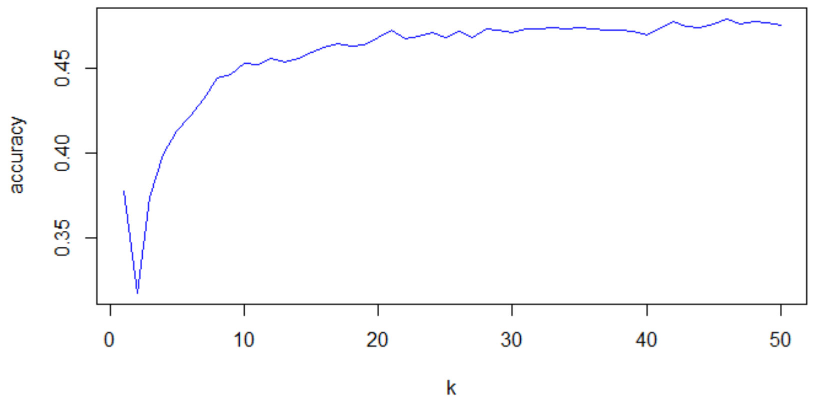
Random forests

The random forests technique gives a very similar result to decision trees with a slightly better 50% accuracy. However, we can see that the model again has the best performance predicting 5 & 1 stars rakings:



K nearest neighbors

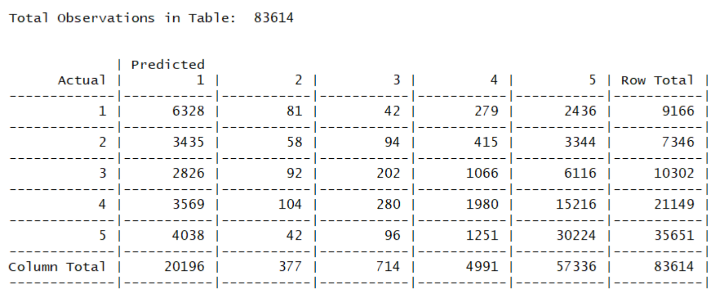
The value of k for this technique was found using a loop that would calculate accuracy for different k values:



The accuracy of this model is 47%, with sensitivity of predicting class 5 is 81%.

Naïve Bayes

The accuracy of this technique is 46%, with sensitivity for class 5 of 85%.

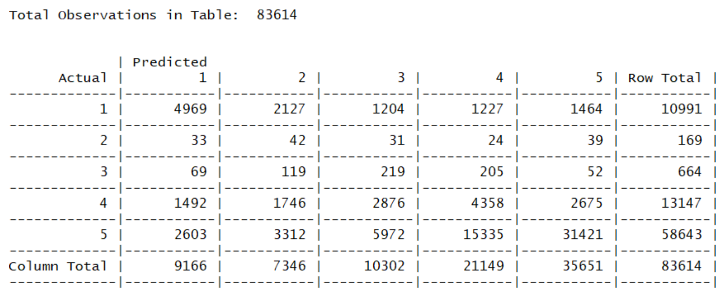


```
Total Observations in Table:  83614 

             | Predicted 
      Actual |         1 |         2 |         3 |         4 |         5 | Row Total | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
           1 |      6328 |        81 |        42 |       279 |      2436 |      9166 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
           2 |      3435 |        58 |        94 |       415 |      3344 |      7346 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
           3 |      2826 |        92 |       202 |      1066 |      6116 |     10302 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
           4 |      3569 |       104 |       280 |      1980 |     15216 |     21149 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
           5 |      4038 |        42 |        96 |      1251 |     30224 |     35651 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
Column Total |     20196 |       377 |       714 |      4991 |     57336 |     83614 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
```

Linear Discriminant Analysis

Accuracy = 49%



```
Total Observations in Table:  83614 

             | Predicted 
      Actual |         1 |         2 |         3 |         4 |         5 | Row Total | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
           1 |      4969 |      2127 |      1204 |      1227 |      1464 |     10991 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
           2 |        33 |        42 |        31 |        24 |        39 |       169 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
           3 |        69 |       119 |       219 |       205 |        52 |       664 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
           4 |      1492 |      1746 |      2876 |      4358 |      2675 |     13147 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
           5 |      2603 |      3312 |      5972 |     15335 |     31421 |     58643 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
Column Total |      9166 |      7346 |     10302 |     21149 |     35651 |     83614 | 
-------------|-----------|-----------|-----------|-----------|-----------|-----------|
```

This method had a similar accuracy that the previous ones, but the sensitivity of predicting a 5 stars ranking is not as good as with the other models.

Approach modification

The previous approach of a 5 class classification model did not show very good results. Random forest was the model that had the best performance, but not good enough to be used in practice. However, the behavior of the error rate for the prediction of classes in several models, suggests that the available data could perform better in a 2 class prediction. Where the objective would change from predicting five classes to predicting two. Furthermore, this change in approach is still aligned with the case study objective, since using this binary prediction would be useful for assisting Yelp to make accurate recommendations to its users.

Binary Classification

The first step in this approach will be to create a new outcome variable called binary, that would have values of 0 or 1. To evaluate the performance we will run the previous models using three different cut-off values for creating the binary outcome variable:

1. Equal to 5 stars = Class 1 & Different from 5 stars = Class 0
2. Bigger or equal to 4 = Class 1 & Lower than 4 = Class 0
3. Bigger or equal to 3 = Class 1 & Lower than 3 = Class 0

Equal to 5 stars & Different from 5 stars

The following table shows the results for the different techniques categorizing 5 stars ranking as positive and lower than 5 stars as negative:

Model	Accuracy	Sensitivity
Random Forest	71%	57%
Logistic regression	71%	61%
Decision trees	71%	55%
K Nearest Neighbors	69%	47%
Naïve Bayes	62%	85%

Predicting for only two classes increased the models accuracy significantly. If we rank the performance of the different techniques, the results are similar to the 5 class classification approach. An interesting result from this is that Naïve Bayes is the technique that has the highest sensitivity to the success class, despite the fact that it has a low accuracy. This means that when the model predicts that a user will give a positive rating to the restaurant, that prediction has a success rate of 85%.

Bigger or equal to 4 stars
The same process was repeated but with a different cutoff value for creating the binary classification. This time a class of 1 was given to all ratings bigger or equal to 4, and a class of 0 to the rest. The following table shows the results:

Model	Accuracy	Sensitivity
Random Forest	76%	92%
Logistic regression	76%	90%
Decision trees	75%	88%
K Nearest Neighbors	74%	92%
Naïve Bayes	74%	88%

Both the models accuracy and sensitivity increased, with random forest being the best one. A 92% sensitivity for a high rating prediction shows that the model has value, and that it could be applied in practice. However, since the accuracy is not high, recommendations will not be possible for all users.

Bigger or equal to 3 stars

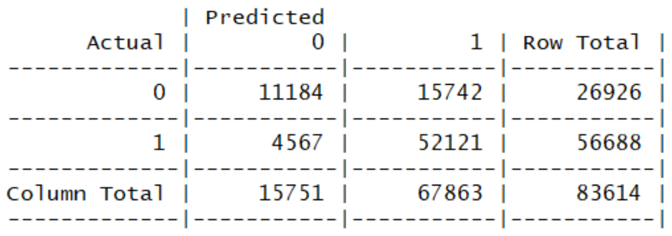
Model	Accuracy	Sensitivity
Random Forest	84%	97%
Logistic regression	85%	96%
Decision trees	84%	96%
K Nearest Neighbors	84%	98%
Naïve Bayes	82%	90%

The model accuracy is now good, but it does not seem to be very useful to predict that a user will give a rating for 3 or more to a restaurant. Moreover, the increase in sensitivity was not very significant compared to the last model.

V. Performance Evaluation

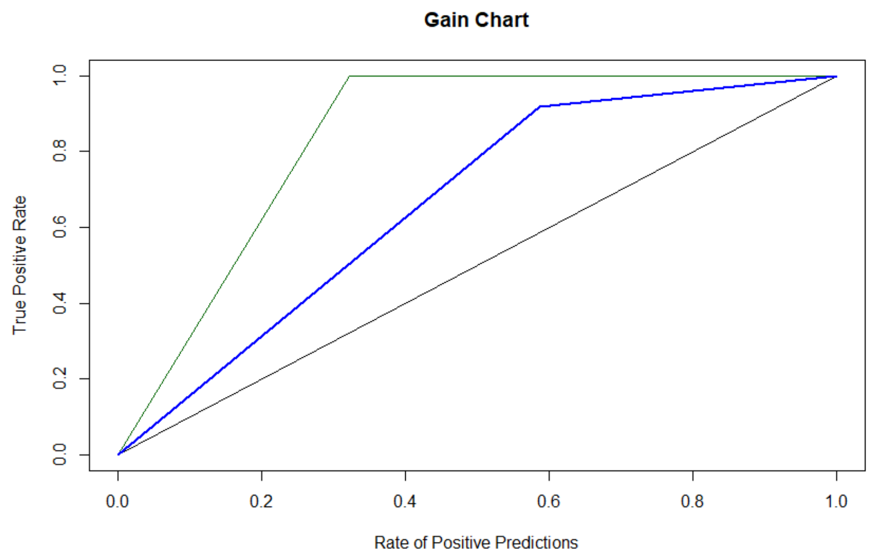
The final result is a Random Forests model that predicts if a user will give a restaurant a ranking between 5 & 4 or between 1 & 3. To evaluate the selected models performance, we will present a confusion matrix and gain chart.

Confusion matrix



```
             | Predicted 
      Actual |         0 |         1 | Row Total | 
-------------|-----------|-----------|-----------|
           0 |     11184 |     15742 |     26926 | 
-------------|-----------|-----------|-----------|
           1 |      4567 |     52121 |     56688 | 
-------------|-----------|-----------|-----------|
Column Total |     15751 |     67863 |     83614 | 
-------------|-----------|-----------|-----------|
```

Accuracy = 76%
Sensitivity = 92%



VI. Discussion and Recommendation

The final result is different from the original problem statement, due to the fact that the available data is not good enough to design a useful model. The decision to change the 5 class prediction to a 2 class prediction proved to be a good decision, with a final model that does a very good job predicting if a user will give a high star rating to a restaurant. However, the model does not have a good performance predicting which restaurants users will rate poorly. The shortcoming of this low accuracy is that the model cannot be used to predict if a user will give a low rating to restaurant. Nevertheless, this problem is not very important, since we are mostly interested in predicting if the users will give high rankings in order to make recommendations. We believe that the main reason for this behavior is the low frequency of low rating stars given by the users, which translates in not enough data to train the model.

A possible solution for increasing the performance, could be to increase the amount of data by adding the actual text review given by the user, using text mining techniques. This part of the reviews data set was not considered in this case study, and could provide much more information about the user's preferences, and increase the accuracy values of the model. Furthermore, the data set had almost 3,000,000 cases, from which we used only 10% of the total data to keep data processing times low. Using a bigger part of this data could also have a positive impact in the final results.

VII. Summary

The original objective of this case study was to create a model that would predict the number of stars a user will give a food business. However, after trying several data mining techniques, we found that the data being used did not provide enough information to make a useful prediction.

The problem and solution of this case study then shifted from a complex approach to a simpler one, and this proved to be a very good solution to generate a useful prediction model from the data. The approach change from a 5 class classification problem to a 2 class classification increased accuracy in 50%, and allowed us to design a model with 92% success rate when predicting that a user will give a high rating to a restaurant.

Several data mining techniques were applied to make this predictions, with Random Forests having the best performance. Furthermore, the application of text mining techniques could probably enhance the results of the model, achieving higher overall accuracy values.

The final result is a very powerful tool that accurately predicts the if a user will give a high rating to a restaurant. This information could later be used by Yelp to make recommendations to users

Appendix: R Code for use case study

Part 1: Merging the data sets

```
library(dplyr)

library(tibble)
library(readxl)

setwd("C:/Users/USER/Google Drive (curiel.m@husky.neu.edu)/Case Study (
1)")

#Read Review Data
reviews1<-as_data_frame(read.csv2("reviews_users1.csv"))

## Warning: `as_data_frame()` is deprecated, use `as_tibble()` (but min
d the new semantics).
## This warning is displayed once per session.

reviews2<-as_data_frame(read.csv2("reviews_users2.csv"))
reviews<-rbind(reviews1, reviews2)
reviews<-reviews[,-1]
colnames(reviews)<-c("user_id", "business_id", "user_stars", "useful",
"funny", "cool")

#Read Business Data
business<-as_data_frame(read_xlsx("Business DB Clean.xlsx"))

reviews2=as_data_frame(merge(reviews,business, by.x ="business_id", by.
y="business_id", all.x = TRUE ))
reviews3<-reviews2%>%arrange(user_id)%>%filter(!is.na(name))

#Read Users Data
users1<-as_data_frame(read_xlsx("Users DB.xlsx"))
users2<-as_data_frame(read_xlsx("User DB pt2.xlsx"))
users<-rbind(users1, users2)


#reviews4=as_data_frame(merge(reviews3,users, by.x ="user_id", by.y="us
er_id", all.x = TRUE ))

#write.csv2(reviews4, file = "Yelp_Dataset.csv")

df<-data.frame(read.csv2("Yelp_Dataset.csv"), stringsAsFactors = FALSE)
df2<-df[,-c(2,3,8,30)]
df2$average_stars<-as.numeric(levels(df2$average_stars))[df2$average_st
ars]
```

Part 2: Creating dummy variables

```
library(dplyr)
library(tibble)
library(readxl)

setwd("C:/Users/USER/Google Drive (curiel.m@husky.neu.edu)/Case Study (
1)")
df2<-data.frame(read.csv2("Yelp_Dataset.csv"))
df2<-df2[,-1]

#Eliminate NA
df2<-df2[complete.cases(df2),]

df2<-df2[,-1]
str(df2)

df2<-as_data_frame(read.csv2("Yelp_Dataset.csv"))

## Warning: `as_data_frame()` is deprecated, use `as_tibble()` (but min
d the new semantics).
## This warning is displayed once per session.

#Create dummy variables
library(fastDummies)
set.seed(100)
#Create dummies
dum<-dummy_cols(df2, select_columns = c("Categories.2", "attributes.Noi
seLevel",
"attributes.RestaurantsPriceRange2","Ambience", "attributes.WiFi", "att
ributes.Alcohol",
"attributes.RestaurantsAttire"), remove_first_dummy = TRUE)
dum2<-dum[,-c(11,15,17,20,21,22,23)]
dum2$user_stars<-as.factor(dum2$user_stars)
colnames(dum2)[37] <- "Categories.2_Fast_Food"
colnames(dum2)[43]<-"Categories.2_Ice_Cream"
colnames(dum2)[41]<-"Categories.2_Middle_East"
colnames(dum2)[40]<-"Categories.2_Non_Alcoholic_Drinks"

#Turn all categorical variables to factors
cols<-c(10:19,27:64)
dum2[cols]<-lapply(dum2[cols], factor)

#Select 10% of total data
set.seed(100) # set seed for reproducing the partition
dum3<-dum2[sample(nrow(dum2), nrow(dum2)*0.1), ]
```

Part 3: Data exploring and PCA

```
#Separate continious variables and evaluate correlation
library(car)

## Loading required package: carData
```

```
cont_df<-dum3[,c(3:9,20:26)]
cont_df1<-dum4[,c(3:9)]
cont_df2<-dum4[,c(20,21,22,23,24,25,26)]
#Evaluate behavior between variables with scatterplotMatrix
scatterplotMatrix(cont_df1, ellipse = FALSE,col=c("black"),
                  regLine = list(method=lm, lty=1, lwd=2, col="red"),
                  smooth=list(smoother=loessLine, spread=FALSE, lty.smo
oth=2, lwd.smooth=1, col.smooth="black"),
                  main="Scatter Plot Matrix")

scatterplotMatrix(cont_df2, ellipse = FALSE,col=c("black"),
                  regLine = list(method=lm, lty=1, lwd=2, col="red"),
                  smooth=list(smoother=loessLine, spread=FALSE, lty.smo
oth=2, lwd.smooth=1, col.smooth="black"),
                  main="Scatter Plot Matrix")

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], lo
g.x =
## FALSE, : could not fit smooth

#The scatterplot show existing correlation between predictors that make
it suitable for PCA dimension reduction
#Evaluate correlation between variables
library(reshape)
library(ggplot2)

cormat<-round(cor(cont_df),2)
melted_cormat<-melt(cormat)
melted_cormat

ggplot(data = melted_cormat, aes(x=X1, y=X2, fill=value)) + geom_tile()
+
  geom_text(aes(X2, X1, label = value), color = "black", size = 4)

#PCA
library(psych)

library(GPArotation)
cont_df<-dum3[,c(3:9,20:26)]
colnames(cont_df)

##  [1] "useful.x"       "funny.x"        "cool.x"         "latitude"
##  [5] "longitude"      "Business_Stars" "review_count.x" "review_coun
t.y"
##  [9] "useful.y"       "funny.y"        "cool.y"         "fans"
## [13] "average_stars"  "compliment_hot"

#Input the raw data matrix to fa.parallel() function to determine the n
umber of components to extract
fa.parallel(cont_df, fa="pc", main = "Scree plot
            with parallel analysis",show.legend=TRUE,ylabel="eigenvalue
s")
```

```
abline(1,0)

pc1<-principal(cont_df, nfactors = 7, rotate = "none" , scores = TRUE)
pc1$loadings

pc<-data.frame(pc1$scores)

#The 5 PC will substitute
#useful.x       funny.x        cool.x         latitude       longitude
#review_count.x review_count.y useful.y       funny.y        cool.y
#fans           compliment_hot   Busieness_Stars    average_stars

pc_dum<-data.frame(dum3[,-c(3:9,20:26)])

pc_dum$PC1<-pc$PC1
pc_dum$PC2<-pc$PC2
pc_dum$PC3<-pc$PC3
pc_dum$PC4<-pc$PC4
pc_dum$PC5<-pc$PC5
pc_dum$PC6<-pc$PC6
pc_dum$PC7<-pc$PC7
```

Part 4: Partition the data and run data mining techniques

```
# partition data, 50% Training, 30% Validation, 20% Test
# Set some input variables to define the splitting.

# Input 2. Set the fractions of the dataframe you want to split into tr
aining,
# validation, and test.
fractionTraining   <- 0.5
fractionValidation <- 0.3
fractionTest       <- 0.2

# Compute sample sizes.
sampleSizeTraining   <- floor(fractionTraining   * nrow(pc_dum))
sampleSizeValidation <- floor(fractionValidation * nrow(pc_dum))
sampleSizeTest       <- floor(fractionTest       * nrow(pc_dum))

# Create the randomly-sampled indices for the dataframe. Use setdiff()
to
# avoid overlapping subsets of indices.
indicesTraining    <- sort(sample(seq_len(nrow(pc_dum)), size=sampleSiz
eTraining))
indicesNotTraining <- setdiff(seq_len(nrow(pc_dum)), indicesTraining)
indicesValidation  <- sort(sample(indicesNotTraining, size=sampleSizeVa
lidation))
indicesTest        <- setdiff(indicesNotTraining, indicesValidation)

# Finally, output the three dataframes for training, validation and tes
t.
```

```
train.df <- pc_dum[indicesTraining, ]
valid.df <- pc_dum[indicesValidation, ]
test.df <- pc_dum[indicesTest, ]

##############################################Classification Tree######
#####
library(rpart)
library(rpart.plot)

class.tree <- rpart(user_stars ~.-X ,data = train.df, method = "class",
cp=0.001)
rpart.plot(class.tree, type =1, digits = 3, fallen.leaves = TRUE)
pruned.tree.class <- prune(class.tree,
cp = class.tree$cptable[which.min(class.tree$cptable[,"xerror"]),"CP"])
rpart.plot(pruned.tree.class, type =1, digits = 3, fallen.leaves = TRUE
)

p1_tree<-predict(pruned.tree.class, train.df)
p1_tree_val<-predict(pruned.tree.class,valid.df)
p1_tree_test<-predict(pruned.tree.class, test.df)

library(caret)

predicted<-as.data.frame(cbind(row.names(p1_tree),apply(p1_tree,1,funct
ion(x)
  colnames(p1_tree)[which(x==max(x))])))
conf.matrix.train<-confusionMatrix(factor(predicted$V2),factor(train.df
$user_stars))

predicted_valid<-as.data.frame(cbind(row.names(p1_tree_val),apply(p1_tr
ee_val,1,function(x)
  colnames(p1_tree_val)[which(x==max(x))])))
conf.matrix.valid<-confusionMatrix(factor(predicted_valid$V2),factor(va
lid.df$user_stars))

CrossTable(x = valid.df[,2], y =predicted_valid$V2, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

predicted_test<-as.data.frame(cbind(row.names(p1_tree_test),apply(p1_tr
ee_test,1,function(x)
  colnames(p1_tree_test)[which(x==max(x))])))
conf.matrix.test<-confusionMatrix(factor(predicted_test$V2),factor(test
.df$user_stars))

###############################################RandomForests###########
######
library(randomForest)

library(ggplot2)
library(cowplot)
```

```
#Reduce data set size for random forest
train.forest<-train.df[sample(nrow(train.df), nrow(train.df)*0.2), ]

rf<-randomForest(user_stars~.-X, data=train.forest, ntree=100, mtry=4,
nodesize=10, importance=TRUE)
oob.error.data <- data.frame(
  Trees=rep(1:nrow(rf$err.rate), times=3),
  Type=rep(c("OOB", "1", "2", "3", "4", "5"), each=nrow(rf$err.rate)),
  Error=c(rf$err.rate[,"OOB"],
          rf$err.rate[,2],
          rf$err.rate[,3],
          rf$err.rate[,4],
          rf$err.rate[,5],
          rf$err.rate[,6]))
ggplot(data=oob.error.data, aes(x=Trees, y=Error)) +geom_line(aes(color
=Type))

rf.valid<-predict(rf, valid.df[,-2])
conf.rf.valid<- confusionMatrix(rf.valid  , factor(valid.df[, 2]))
CrossTable(x = valid.df[,2], y =rf.valid, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

##########################################################knn##############
######
library(caret)
library(FNN)

train.knn<-train.df[sample(nrow(train.df), nrow(train.df)*0.05), ]
valid.knn<-valid.df[sample(nrow(valid.df), nrow(valid.df)*0.05), ]

training<-model.matrix(~.-user_stars-X, data = train.knn)
validation<-model.matrix(~.-user_stars-X, data=valid.knn)

kn<-knn(training, test = validation, cl=train.knn[,2], k=1)
conf.matrix.kn<- confusionMatrix(kn, factor(valid.knn[, 2]))

accuracy.df<-data.frame(k=seq(1,50,1),accuracy=rep(0,50))

for(i in 1:50){
  knn2 <- knn(training, test=validation,cl = train.knn[,2], k = i)
accuracy.df[i,2]<-confusionMatrix(knn2, factor(valid.knn[,2]))$overall[
1]
}

plot(accuracy.df, type="line", col="blue")

#best knn is k = 30

kn.train<-knn(training, test = validation, cl=train.knn[,2], k=30)
conf.kn.train<- confusionMatrix(kn.train, factor(valid.knn[, 2]))
```

```
CrossTable(x = valid.knn[,2], y =kn.train, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

#####################################NaiveBayes#######################
######
library(e1071)

nb_model<-naiveBayes(user_stars~.-X, data = train.df)
#Test the model using the validation data
nb_model_train<-predict(nb_model, train.df[,-2])
#Confusion matrix
confussion.train<-confusionMatrix(nb_model_train, train.df$user_stars)$
overall[1]
CrossTable(x = nb_model_train, y =train.df$user_stars, prop.chisq =FALS
E,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

nb_model_valid<-predict(nb_model, valid.df[,-2])
#Confusion matrix
confussion.valid<-confusionMatrix(nb_model_valid, valid.df$user_stars)
CrossTable(x =valid.df$user_stars , y =nb_model_valid, prop.chisq =FALS
E,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

#####################################################LDA###############
######
library(MASS)
lda <- lda(user_stars~.-X, train.df)
lda.predict.train <- predict(lda, train.df[,-2])
lda.confusion.train<-confusionMatrix(lda.predict.train$class, train.df$
user_stars)
CrossTable(x = train.df$user_stars, y =lda.predict.train$class, prop.ch
isq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

lda.predict.valid <- predict(lda, valid.df[,-2])
lda.confusion.valid<-confusionMatrix(lda.predict.valid$class, valid.df$
user_stars)
CrossTable(x =lda.predict.valid$class , y =valid.df$user_stars, prop.ch
isq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))
```

Part 5: Binary classification approach

```
#BINARY CLASSIFICATION APPROACH
#Binary approach
```

```
##############################################Equal to
5#######################
#Create binary stars classification
train.df$binary<-ifelse(as.numeric(train.df$user_stars)==5,1,0)
train.df$binary<-as.factor(train.df$binary)
valid.df$binary<-ifelse(as.numeric(valid.df$user_stars)==5,1,0)
valid.df$binary<-as.factor(valid.df$binary)

#Logistic regression

pred_log<-glm(binary~.-user_stars-X, data = train.df, family =
"binomial")
log_pred<-predict(pred_log, valid.df[,-58], type="response")

predict<-ifelse(log_pred>0.5,1,0)
CrossTable(x =valid.df$binary , y =predict, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn =
c("Actual", "Predicted"))
#Random Forest
library(randomForest)
train.forest<-train.df[sample(nrow(train.df), nrow(train.df)*0.2), ]

rf<-randomForest(binary~.-X-user_stars, data=train.forest, ntree=100, m
try=4, nodesize=10, importance=TRUE)
oob.error.data <- data.frame(
  Trees=rep(1:nrow(rf$err.rate), times=3),
  Type=rep(c("OOB", "Negative rating", "Positive rating"), each=nrow(rf
$err.rate)),
  Error=c(rf$err.rate[,"OOB"],
          rf$err.rate[,2],
          rf$err.rate[,3]))
ggplot(data=oob.error.data, aes(x=Trees, y=Error)) +geom_line(aes(color
=Type))

#100 forests is enough

colnames(valid.df)

rf.valid<-predict(rf, valid.df[,-58])
conf.rf.valid<- confusionMatrix(rf.valid  , factor(valid.df[, 58]))
CrossTable(x = valid.df[,2], y =rf.valid, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

#Decision Trees
train.df2<-train.df[,-2]
valid.df2<-valid.df[,-2]
class.tree <- rpart(binary ~.-X ,data = train.df2, method = "class", cp
=0.001)
pruned.tree.class <- prune(class.tree,
                           cp = class.tree$cptable[which.min(class.tree
$cptable[,"xerror"]),"CP"])
p1_tree_val<-predict(pruned.tree.class,valid.df2)

predicted_valid<-as.data.frame(cbind(row.names(p1_tree_val),apply(p1_tr
ee_val,1,function(x)
  colnames(p1_tree_val)[which(x==max(x))])))
conf.matrix.valid<-confusionMatrix(factor(predicted_valid$V2),factor(va
lid.df2$binary))

##################knn

library(FNN)

train.knn<-train.df[sample(nrow(train.df), nrow(train.df)*0.05), ]
valid.knn<-valid.df[sample(nrow(valid.df), nrow(valid.df)*0.05), ]

training<-model.matrix(~.-user_stars-X-binary, data = train.knn)
validation<-model.matrix(~.-user_stars-X-binary, data=valid.knn)

kn<-knn(training, test = validation, cl=train.knn[,58], k=30)
conf.matrix.kn<- confusionMatrix(kn, factor(valid.knn[, 58]))

#######################Naive Bayes
library(e1071)
colnames(train.df2)

nb_model<-naiveBayes(binary~.-X, data = train.df2)
#Test the model using the validation data
nb_model_valid<-predict(nb_model, valid.df2[,-57])

#Confusion matrix
confussion.valid<-confusionMatrix(nb_model_valid, valid.df2$binary)
CrossTable(x =valid.df$binary , y =nb_model_valid, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

# Input 2. Set the fractions of the dataframe you want to split into tr
aining,
# validation, and test.
fractionTraining   <- 0.5
fractionValidation <- 0.3
fractionTest       <- 0.2

# Compute sample sizes.
sampleSizeTraining   <- floor(fractionTraining   * nrow(pc_dum))
sampleSizeValidation <- floor(fractionValidation * nrow(pc_dum))
sampleSizeTest       <- floor(fractionTest       * nrow(pc_dum))

# Create the randomly-sampled indices for the dataframe. Use setdiff()
```

```
to
# avoid overlapping subsets of indices.
indicesTraining    <- sort(sample(seq_len(nrow(pc_dum)), size=sampleSiz
eTraining))
indicesNotTraining <- setdiff(seq_len(nrow(pc_dum)), indicesTraining)
indicesValidation  <- sort(sample(indicesNotTraining, size=sampleSizeVa
lidation))
indicesTest        <- setdiff(indicesNotTraining, indicesValidation)

# Finally, output the three dataframes for training, validation and tes
t.
train.df <- pc_dum[indicesTraining, ]
valid.df <- pc_dum[indicesValidation, ]
test.df <- pc_dum[indicesTest, ]

############################Bigger or equal to 4########################
####
#Create binary stars classification
train.df$binary<-ifelse(as.numeric(train.df$user_stars)>=4,1,0)
train.df$binary<-as.factor(train.df$binary)
valid.df$binary<-ifelse(as.numeric(valid.df$user_stars)>=4,1,0)
valid.df$binary<-as.factor(valid.df$binary)

#Logistic regression

pred_log<-glm(binary~.-user_stars-X, data = train.df, family = "binomia
l")
log_pred<-predict(pred_log, valid.df[,-58], type="response")

predict<-ifelse(log_pred>0.5,1,0)
CrossTable(x =valid.df$binary , y =predict, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

conf.log.valid<- confusionMatrix(factor(predict), factor(valid.df$binar
y))


#Random Forest
train.forest<-train.df[sample(nrow(train.df), nrow(train.df)*0.2), ]

rf<-randomForest(binary~.-X-user_stars, data=train.forest, ntree=100, m
try=4, nodesize=10, importance=TRUE)
oob.error.data <- data.frame(
  Trees=rep(1:nrow(rf$err.rate), times=3),
  Type=rep(c("OOB", "Negative rating", "Positive rating"), each=nrow(rf
$err.rate)),
  Error=c(rf$err.rate[,"OOB"],
          rf$err.rate[,2],
          rf$err.rate[,3]))
```

```
ggplot(data=oob.error.data, aes(x=Trees, y=Error)) +geom_line(aes(color
=Type))

#100 forests is enough

rf.valid<-predict(rf, valid.df[,-58])
conf.rf.valid<- confusionMatrix(rf.valid  , factor(valid.df[, 58]))
CrossTable(x = valid.df[,2], y =rf.valid, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

#Decision Trees
train.df2<-train.df[,-2]
valid.df2<-valid.df[,-2]
class.tree <- rpart(binary ~.-X ,data = train.df2, method = "class", cp
=0.001)
pruned.tree.class <- prune(class.tree,
                           cp = class.tree$cptable[which.min(class.tree
$cptable[,"xerror"]),"CP"])
p1_tree_val<-predict(pruned.tree.class,valid.df2)

predicted_valid<-as.data.frame(cbind(row.names(p1_tree_val),apply(p1_tr
ee_val,1,function(x)
  colnames(p1_tree_val)[which(x==max(x))])))
conf.matrix.valid<-confusionMatrix(factor(predicted_valid$V2),factor(va
lid.df2$binary))

##################knn

train.knn<-train.df[sample(nrow(train.df), nrow(train.df)*0.05), ]
valid.knn<-valid.df[sample(nrow(valid.df), nrow(valid.df)*0.05), ]

training<-model.matrix(~.-user_stars-X-binary, data = train.knn)
validation<-model.matrix(~.-user_stars-X-binary, data=valid.knn)

kn<-knn(training, test = validation, cl=train.knn[,58], k=30)
conf.matrix.kn<- confusionMatrix(kn, factor(valid.knn[, 58]))

#######################Naive Bayes
nb_model<-naiveBayes(binary~.-X, data = train.df2)
#Test the model using the validation data
nb_model_valid<-predict(nb_model, valid.df2[,-57])

#Confusion matrix
confussion.valid<-confusionMatrix(nb_model_valid, valid.df2$binary)
CrossTable(x =valid.df$binary , y =nb_model_valid, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))
```

```
########################################################################
######

# Input 2. Set the fractions of the dataframe you want to split into tr
aining,
# validation, and test.
fractionTraining   <- 0.5
fractionValidation <- 0.3
fractionTest       <- 0.2

# Compute sample sizes.
sampleSizeTraining   <- floor(fractionTraining   * nrow(pc_dum))
sampleSizeValidation <- floor(fractionValidation * nrow(pc_dum))
sampleSizeTest       <- floor(fractionTest       * nrow(pc_dum))

# Create the randomly-sampled indices for the dataframe. Use setdiff()
to
# avoid overlapping subsets of indices.
indicesTraining    <- sort(sample(seq_len(nrow(pc_dum)), size=sampleSiz
eTraining))
indicesNotTraining <- setdiff(seq_len(nrow(pc_dum)), indicesTraining)
indicesValidation  <- sort(sample(indicesNotTraining, size=sampleSizeVa
lidation))
indicesTest        <- setdiff(indicesNotTraining, indicesValidation)

# Finally, output the three dataframes for training, validation and tes
t.
train.df <- pc_dum[indicesTraining, ]
valid.df <- pc_dum[indicesValidation, ]
test.df <- pc_dum[indicesTest, ]

#############################Bigger or equal to 3######################
#####
#Create binary stars classification
train.df$binary<-ifelse(as.numeric(train.df$user_stars)>=3,1,0)
train.df$binary<-as.factor(train.df$binary)
valid.df$binary<-ifelse(as.numeric(valid.df$user_stars)>=3,1,0)
valid.df$binary<-as.factor(valid.df$binary)

#Logistic regression

pred_log<-glm(binary~.-user_stars-X, data = train.df, family = "binomia
l")
log_pred<-predict(pred_log, valid.df[,-58], type="response")

predict<-ifelse(log_pred>0.5,1,0)
CrossTable(x =valid.df$binary , y =predict, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))
```

```
conf.log.valid<- confusionMatrix(factor(predict), factor(valid.df$binar
y))


#Random Forest
train.forest<-train.df[sample(nrow(train.df), nrow(train.df)*0.2), ]

rf<-randomForest(binary~.-X-user_stars, data=train.forest, ntree=100, m
try=4, nodesize=10, importance=TRUE)
oob.error.data <- data.frame(
  Trees=rep(1:nrow(rf$err.rate), times=3),
  Type=rep(c("OOB", "Negative rating", "Positive rating"), each=nrow(rf
$err.rate)),
  Error=c(rf$err.rate[,"OOB"],
          rf$err.rate[,2],
          rf$err.rate[,3]))
ggplot(data=oob.error.data, aes(x=Trees, y=Error)) +geom_line(aes(color
=Type))

#100 forests is enough

rf.valid<-predict(rf, valid.df[,-58])
conf.rf.valid<- confusionMatrix(rf.valid  , factor(valid.df[, 58]))
CrossTable(x = valid.df[,58], y =rf.valid, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

#Decision Trees
train.df2<-train.df[,-2]
valid.df2<-valid.df[,-2]
class.tree <- rpart(binary ~.-X ,data = train.df2, method = "class", cp
=0.001)
pruned.tree.class <- prune(class.tree,
                           cp = class.tree$cptable[which.min(class.tree
$cptable[,"xerror"]),"CP"])
p1_tree_val<-predict(pruned.tree.class,valid.df2)

predicted_valid<-as.data.frame(cbind(row.names(p1_tree_val),apply(p1_tr
ee_val,1,function(x)
  colnames(p1_tree_val)[which(x==max(x))])))
conf.matrix.valid<-confusionMatrix(factor(predicted_valid$V2),factor(va
lid.df2$binary))

##################knn

train.knn<-train.df[sample(nrow(train.df), nrow(train.df)*0.05), ]
valid.knn<-valid.df[sample(nrow(valid.df), nrow(valid.df)*0.05), ]

training<-model.matrix(~.-user_stars-X-binary, data = train.knn)
validation<-model.matrix(~.-user_stars-X-binary, data=valid.knn)
```

```
kn<-knn(training, test = validation, cl=train.knn[,58], k=30)
conf.matrix.kn<- confusionMatrix(kn, factor(valid.knn[, 58]))

########################Naive Bayes
nb_model<-naiveBayes(binary~.-X, data = train.df2)
#Test the model using the validation data
nb_model_valid<-predict(nb_model, valid.df2[,-57])

#Confusion matrix
confussion.valid<-confusionMatrix(nb_model_valid, valid.df2$binary)
CrossTable(x =valid.df$binary , y =nb_model_valid, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))
```

Part 6: Final evaluation

```
# Input 2. Set the fractions of the dataframe you want to split into
training,
# validation, and test.
fractionTraining   <- 0.5
fractionValidation <- 0.3
fractionTest       <- 0.2

# Compute sample sizes.
sampleSizeTraining   <- floor(fractionTraining   * nrow(pc_dum))
sampleSizeValidation <- floor(fractionValidation * nrow(pc_dum))
sampleSizeTest       <- floor(fractionTest       * nrow(pc_dum))

# Create the randomly-sampled indices for the dataframe. Use setdiff()
to
# avoid overlapping subsets of indices.
indicesTraining    <- sort(sample(seq_len(nrow(pc_dum)),
size=sampleSizeTraining))
indicesNotTraining <- setdiff(seq_len(nrow(pc_dum)), indicesTraining)
indicesValidation  <- sort(sample(indicesNotTraining,
size=sampleSizeValidation))
indicesTest        <- setdiff(indicesNotTraining, indicesValidation)

# Finally, output the three dataframes for training, validation and
test.
train.df <- pc_dum[indicesTraining, ]
valid.df <- pc_dum[indicesValidation, ]
test.df <- pc_dum[indicesTest, ]

#Create binary stars classification
train.df$binary<-ifelse(as.numeric(train.df$user_stars)>=4,1,0)
train.df$binary<-as.factor(train.df$binary)
```

```
valid.df$binary<-ifelse(as.numeric(valid.df$user_stars)>=4,1,0)
valid.df$binary<-as.factor(valid.df$binary)

#Random Forest
train.forest<-train.df[sample(nrow(train.df), nrow(train.df)*0.2), ]

rf<-randomForest(binary~.-X-user_stars, data=train.forest, ntree=100,
mtry=4, nodesize=10, importance=TRUE)
oob.error.data <- data.frame(
  Trees=rep(1:nrow(rf$err.rate), times=3),
  Type=rep(c("OOB", "Negative rating", "Positive rating"),
each=nrow(rf$err.rate)),
  Error=c(rf$err.rate[,"OOB"],
          rf$err.rate[,2],
          rf$err.rate[,3]))
ggplot(data=oob.error.data, aes(x=Trees, y=Error))
+geom_line(aes(color=Type))
#100 forests is enough

rf.valid<-predict(rf, valid.df[,-58], type="class")

conf.rf.valid<- confusionMatrix(rf.valid  , factor(valid.df[, 58]))
CrossTable(x = valid.df[,58], y =rf.valid, prop.chisq =FALSE,
           prop.c = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c("Act
ual", "Predicted"))

library(ROCR)

v<-c(pred=rf.valid)
pred<-prediction(v,valid.df[,58])
gain<-performance(pred,"tpr","fpr" )
proportion<-data.frame(table(valid.df$binary))
cutoff<-proportion$Freq[1]/(proportion$Freq[2]+proportion$Freq[1])
plot(x=c(0, 1), y=c(0, 1), type="l", col="black", lwd=1,
     ylab="True Positive Rate",
     xlab="Rate of Positive Predictions", main="Gain Chart")
lines(x=c(0, cutoff, 1), y=c(0, 1, 1), col="darkgreen", lwd=1)
gain.x = unlist(slot(gain, 'x.values'))
gain.y = unlist(slot(gain, 'y.values'))
lines(x=gain.x, y=gain.y, col="blue", lwd=2)
```