SUVARNA JOSHI (hf2351)
NALISHA RATHOD (gs2440)

Las Vegas

Hotel Recommendation System

CSC 5800 Intelligent Systems

Course Project

December 07, 2020

Table of Contents

Overview of the problem

The hotel industry has a variety of applications/websites who provide different services to their clients. The data on the customer experience and the ratings of the hotels can be used to develop a recommendation to a client. Here in, we undertake to analyze the user characteristics and experience in addition to the hotel services and ratings of the hotel and then predict the overall score the hotel can receive. The score can be used to provide recommendation to the customer that needs a hotel, based on his/her preferences and attributes and the hotel information.

Dataset overview

We use the **Las Vegas Strip Data Set** data set that was obtained from the UCI Machine Learning Repository. According to the data source, the data contains 504 instances, 20attributes and is fit for undertaking classification and regression tasks.

Data Description

Loading up the required libraries for description of the data.

```
library(tidyverse)
library(readr)
```

Loading up the data using the readr::read_delim() function that best captures the delimiter of the data. The data can best be separated and described based on the user and the facilities and ratings of the hotel.

```
lasVegas <- read_delim("LasVegasTripAdvisorReviews-Dataset.csv",
                       ";", escape_double = FALSE, trim_ws = TRUE)
#dimensions of the data
dim(lasVegas)

## [1] 504  20

#Hotel information
knitr::kable( head(lasVegas[, c(8:13,15:16)]),
            caption = "Hotel Attributes")
```

Hotel Attributes

Pool	Gym	Tennis court	Spa	Casino	Free internet	Hotel stars	Nr. rooms
NO	YES	NO	NO	YES	YES	3	3773
NO	YES	NO	NO	YES	YES	3	3773
NO	YES	NO	NO	YES	YES	3	3773
NO	YES	NO	NO	YES	YES	3	3773
NO	YES	NO	NO	YES	YES	3	3773
NO	YES	NO	NO	YES	YES	3	3773

```r
#User information
knitr::kable( head(lasVegas[,c(1:7,17,18)]),
          caption = "User Attributes" )
```

User Attributes

User country	Nr. reviews	Nr. hotel reviews	Helpful votes	Score	Period of stay	Traveler type	User continent	Member years
USA	11	4	13	5	Dec-Feb	Friends	North America	9
USA	119	21	75	3	Dec-Feb	Business	North America	3
USA	36	9	25	5	Mar-May	Families	North America	2
UK	14	7	14	4	Mar-May	Friends	Europe	6
Canada	5	5	2	4	Mar-May	Solo	North America	7
Canada	31	8	27	3	Mar-May	Couples	North America	2

```r
knitr::kable(table(lasVegas$`Traveler type`), caption = "Traveler Type")
```

Traveler Type

Var1	Freq
Business	74
Couples	214
Families	110
Friends	82
Solo	24

We then check for missing missing and inconsistent values within our data.

```r
#no missing data
sum(is.na(lasVegas))

## [1] 0
```

The data has no missing values within.

```r
#Checking unique values
unique(lasVegas$`Member years`)# "Unique `Member years` values"

##  [1]     9     3     2     6     7     4     0     5     1    10    11     8
## [13] -1806    12    13
```

```r
#Inconsistent values
knitr::kable( lasVegas[lasVegas$`Member years`==-1806,c(1:5,18)],
          caption = "Inconsistent values")
```

Inconsistent values

User country	Nr. reviews	Nr. hotel reviews	Helpful votes	Score	Member years
USA	17	9	16	5	-1806

The data contains a rather peculiar value in the midst of the Member years column of -1806 that we can assume may be an input error or may be just a dummy for missing value. However, proceeding on we shall exclude it when undertaking algorithm analysis.

Visualization

Count of hotel users by country

```r
lasVegas %>%
  group_by(`User country`) %>%
  summarise(n = n()) %>%
  mutate( percent = n/sum(n)*100)  %>%
  arrange(desc(n)) %>%
  slice_max(n, n =5)
```

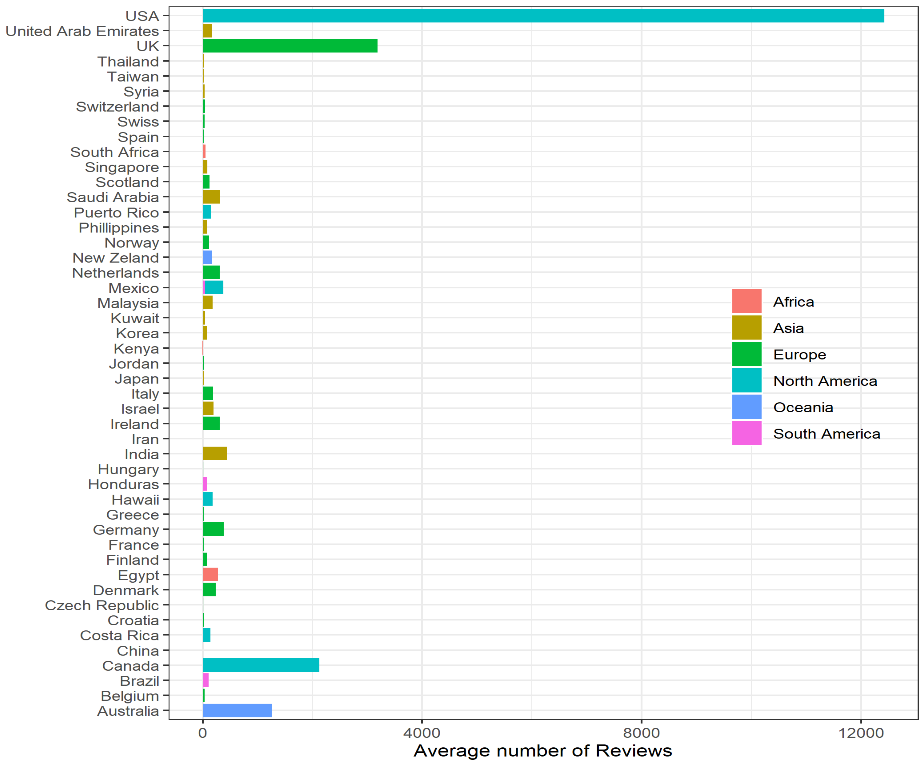
```
## # A tibble: 5 x 3
##   `User country`     n percent
##   <chr>          <int>   <dbl>
## 1 USA              217   43.1
## 2 UK                72   14.3
## 3 Canada            65   12.9
## 4 Australia         36    7.14
## 5 Ireland           13    2.58
```

Majority of the hotel users are based in the USA, that forms 43% of the total user counts per country.

Average ratings

To find the average ratings per country, we group the data by country and plot.

```r
lasVegas %>%
  group_by(`User country`) %>%
  summarise(`User continent` = `User continent`,
            total_review = sum(`Nr. reviews`),
            average_review = mean(`Nr. reviews`)) %>%
  arrange(desc(total_review)) %>%
  ggplot(aes(y = `User country`, x = average_review,
             fill = `User continent`))+
  geom_col() +  labs(x = "Average number of Reviews") +
  theme_bw() +
  theme(axis.title.y = element_blank(),
        legend.position = c(.85,.5),
        legend.title = element_blank())+
  theme(legend.background =  element_blank())
```



The users from USA has the higher average number of reviews for the hotels. Other countries have low review averages.

The average scores per country.

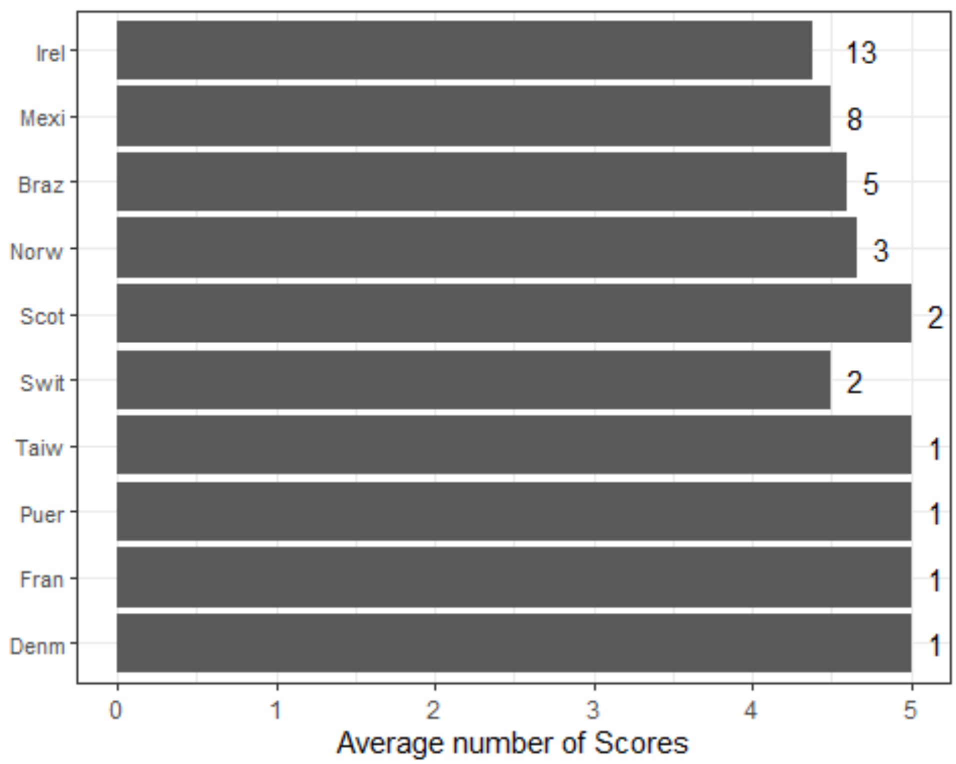
```
lasVegas %>%
  group_by(`User country`) %>%
  summarise(total_score = sum(Score),
          average_score = mean(Score),
          n = n()) %>%
  arrange(desc(average_score)) %>%
  slice_max(average_score, n =5) %>%
  knitr::kable(caption = "Countries with higher scores for the hotels")
```

Countries with higher scores for the hotels

User country	total_score	average_score	N
Denmark	5	5	1
France	5	5	1
Puerto Rico	5	5	1
Scotland	10	5	2
Taiwan	5	5	1

```
#average of scores for hotels by the country of user and count of users
lasVegas %>%
  group_by(`User country`) %>%
```

```r
  summarise(total_score = sum(Score),
            average_score = mean(Score),
            n = n()) %>%
  slice_max(average_score, n = 10) %>%
  mutate(country = if (length(`User country`) <= 3) {`User country`}
         else {str_sub(`User country`,1,4)}) %>%
  ggplot(aes(x = reorder(country, total_score), y = average_score)) +
  geom_col() + labs(y= "Average number of Scores") +
  coord_flip() +

  geom_text(aes(label = n, hjust = -1)) +
  theme_bw() +
  theme(axis.title.y = element_blank(),
       axis.text.y = element_text(size = 8))
```

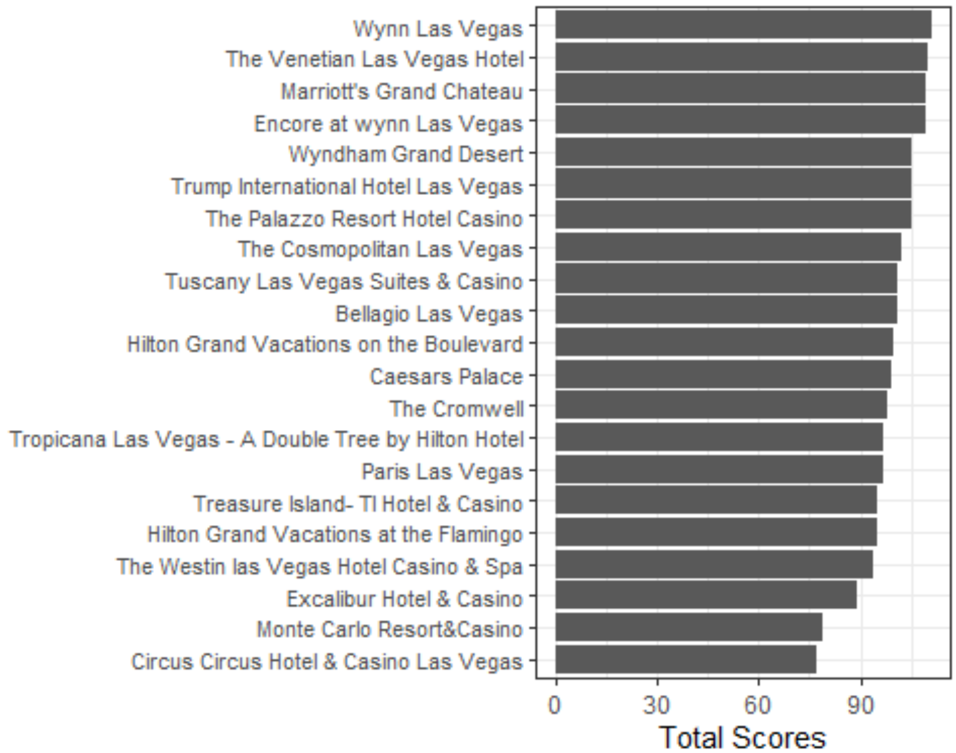


Average Hotel Scores

Score votes by hotel

```r
#total Score for hotels
lasVegas %>%
  group_by(`Hotel name`) %>%
  summarise(total_score = sum(Score),
            rating  = mean("Hotel stars")) %>%
  ggplot(aes(y = reorder(`Hotel name`, total_score), x = total_score)) +
  geom_col() + labs(x = "Total Scores") +
```

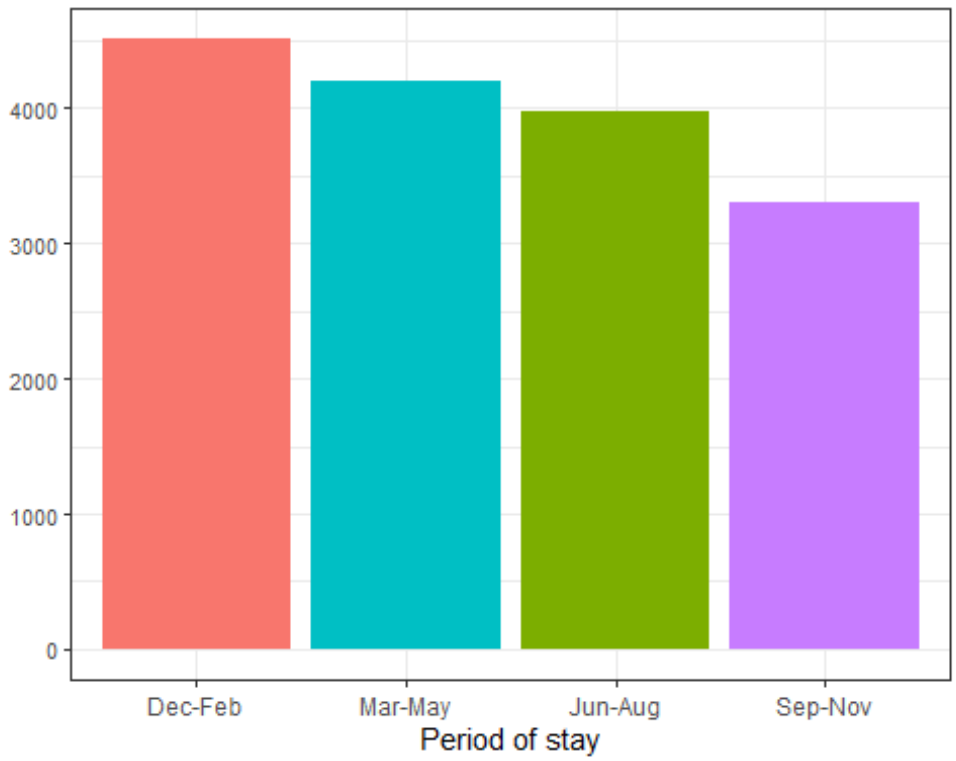
```
  theme_bw() +
  theme(axis.title.y = element_blank(),
        axis.text.y = element_text(size = 8))
```



Total Hotel Scores

Votes by period of Stay

```
#total helpful votes for hotels by the stay
lasVegas %>%
  group_by(`Period of stay`) %>%
  summarise(total_votes = sum(`Helpful votes`)) %>%
  ggplot(aes(x = `Period of stay`, y = total_votes, fill = `Period of stay`)) +
  geom_col() + labs(y= "Total Votes") +
  scale_x_discrete(limits = c("Dec-Feb","Mar-May","Jun-Aug","Sep-Nov")) +
  theme_bw() + guides(fill = FALSE) +
  theme(axis.title.y = element_blank(),
        axis.text.y = element_text(size = 8))
```

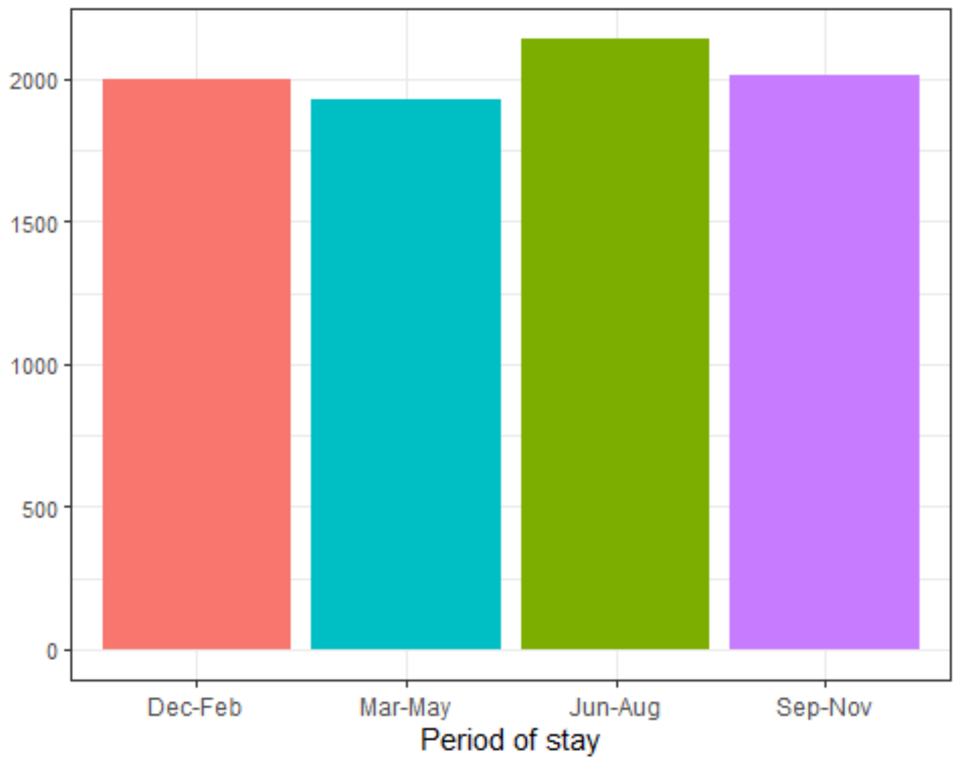


Votes by Period of Stay

The votes per period of time reduce over the course of the year.

Hotel reviews by stay

```
#total helpful votes for hotels by the stay
lasVegas %>%
  group_by(`Period of stay`) %>%
  summarise(total_review = sum(`Nr. hotel reviews`)) %>%
  ggplot(aes(x = `Period of stay`, y = total_review, fill = `Period of stay`)) +
  geom_col() + labs(y= "Total Hotel Reviews") +
  scale_x_discrete(limits = c("Dec-Feb","Mar-May","Jun-Aug","Sep-Nov")) +
  theme_bw() + guides(fill = FALSE) +
  theme(axis.title.y = element_blank(),
        axis.text.y = element_text(size = 8))
```



Reviews by Period of Stay

The review are highest in the period Jun-Aug and helpful votes in Dec-Feb.

Algorithms

Data Preparation

We filter out the unnecessary data for undertaking our analysis.

```
lasVegas <-
  lasVegas %>%
  filter(!grepl("-1806",`Member years`)) %>%
  select(-`Nr. reviews`,-`Nr. hotel reviews`,-`Helpful votes`, -`Hotel name`,
         -`Review month`,-`Review weekday`)
```

Conversion of data Points to factor values

```
str(lasVegas)

## tibble [503 x 14] (S3: tbl_df/tbl/data.frame)
##  $ User country  : chr [1:503] "USA" "USA" "USA" "UK" ...
##  $ Score         : num [1:503] 5 3 5 4 4 3 4 4 4 3 ...
##  $ Period of stay: chr [1:503] "Dec-Feb" "Dec-Feb" "Mar-May" "Mar-May" ...
##  $ Traveler type : chr [1:503] "Friends" "Business" "Families" "Friends" ...
##  $ Pool          : chr [1:503] "NO" "NO" "NO" "NO" ...
##  $ Gym           : chr [1:503] "YES" "YES" "YES" "YES" ...
##  $ Tennis court  : chr [1:503] "NO" "NO" "NO" "NO" ...
##  $ Spa           : chr [1:503] "NO" "NO" "NO" "NO" ...
##  $ Casino        : chr [1:503] "YES" "YES" "YES" "YES" ...
##  $ Free internet : chr [1:503] "YES" "YES" "YES" "YES" ...
##  $ Hotel stars   : num [1:503] 3 3 3 3 3 3 3 3 3 3 ...
##  $ Nr. rooms     : num [1:503] 3773 3773 3773 3773 3773 ...
##  $ User continent: chr [1:503] "North America" "North America" "North America" "Europe" .
..
##  $ Member years  : num [1:503] 9 3 2 6 7 2 4 0 3 5 ...

lasVegas$`User country` <- as.factor(lasVegas$`User country`)
lasVegas$`Period of stay` <- as.factor(lasVegas$`Period of stay`)
lasVegas$`Traveler type` <- as.factor(lasVegas$`Traveler type`)
lasVegas$Pool <- as.factor(lasVegas$Pool)
lasVegas$Gym <- as.factor(lasVegas$Gym)
lasVegas$`Tennis court` <- as.factor(lasVegas$`Tennis court`)
lasVegas$Spa <- as.factor(lasVegas$Spa)
lasVegas$Casino <- as.factor(lasVegas$Casino)
lasVegas$`Free internet` <- as.factor(lasVegas$`Free internet`)
lasVegas$`User continent` <- as.factor(lasVegas$`User continent`)
lasVegas$Score <- as.factor(lasVegas$Score)
lasVegas$`Nr. rooms` <- as.factor(lasVegas$`Nr. rooms`)
lasVegas$`Hotel stars` <- as.factor(lasVegas$`Hotel stars`)
str(lasVegas)

## tibble [503 x 14] (S3: tbl_df/tbl/data.frame)
##  $ User country  : Factor w/ 47 levels "Australia","Belgium",..: 47 47 47 45 4 4 45 47 18 4 ...
##  $ Score         : Factor w/ 5 levels "1","2","3","4",..: 5 3 5 4 4 3 4 4 4 3 ...
##  $ Period of stay: Factor w/ 4 levels "Dec-Feb","Jun-Aug",..: 1 1 3 3 3 3 3 3 3 3 ...
##  $ Traveler type : Factor w/ 5 levels "Business","Couples",..: 4 1 3 4 5 2 2 3 4 3 ...
##  $ Pool          : Factor w/ 2 levels "NO","YES": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Gym           : Factor w/ 2 levels "NO","YES": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Tennis court  : Factor w/ 2 levels "NO","YES": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Spa           : Factor w/ 2 levels "NO","YES": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Casino        : Factor w/ 2 levels "NO","YES": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Free internet : Factor w/ 2 levels "NO","YES": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Hotel stars   : Factor w/ 5 levels "3","4","5","35",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
##  $ Nr. rooms      : Factor w/ 21 levels "188","315","716",..: 18 18 18 18 18 18 18 18 18 18 ...
##  $ User continent: Factor w/ 6 levels "Africa","Asia",..: 4 4 4 3 4 4 3 4 2 4 ...
##  $ Member years  : num [1:503] 9 3 2 6 7 2 4 0 3 5 ...
```

Algorithm Selection

setting seed to capture randomness.

```
set.seed(1)
```

Naive Bayes

Loading up of the required library and partitioning into train (60%) and validation (40%) data. We use the naive bayes probabilistic classifier to be able to place a hotel on either scale of the score, with the highest score of 5 being recommendation of a better hotel.

Implementation

```
library(e1071)
#Naive Bayes
train_index <- sample(c(1:dim(lasVegas)[1]), dim(lasVegas)[1]*0.6)
train_LA <- lasVegas[train_index, ]
valid_LA <- lasVegas[-train_index,]
```

Running of the naive bayes algorithm. The naive bayes works well with categorical variables and makes it well suited for our data. The score is our value of interest, with each row of the data as a separate observation. We need to predict the score of the hotel based on the attributes, and the higher the score, the highly recommended it will be.

```r
# run naive bayes
lA_nb <- naiveBayes(Score ~ ., data = train_LA)
lA_nb

##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##          1          2          3          4          5
## 0.01993355 0.05315615 0.14950166 0.30897010 0.46843854
##
## Conditional probabilities:
##    User country
## Y     Australia     Belgium      Brazil      Canada       China  Costa Rica
##   1 0.000000000 0.000000000 0.000000000 0.000000000 0.166666667 0.000000000
##   2 0.000000000 0.000000000 0.000000000 0.250000000 0.000000000 0.000000000
##   3 0.044444444 0.022222222 0.000000000 0.133333333 0.000000000 0.000000000
##   4 0.096774194 0.000000000 0.000000000 0.118279570 0.000000000 0.000000000
##   5 0.078014184 0.000000000 0.028368794 0.127659574 0.000000000 0.007092199
##    User country
## Y       Croatia Czech Republic     Denmark       Egypt     Finland      France
##   1 0.000000000    0.000000000 0.000000000 0.166666667 0.000000000 0.000000000
##   2 0.000000000    0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   3 0.022222222    0.000000000 0.000000000 0.000000000 0.022222222 0.000000000
##   4 0.000000000    0.010752688 0.000000000 0.010752688 0.010752688 0.000000000
##   5 0.000000000    0.000000000 0.000000000 0.000000000 0.007092199 0.007092199
##    User country
## Y       Germany      Greece      Hawaii    Honduras     Hungary       India
##   1 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   2 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   3 0.000000000 0.000000000 0.022222222 0.000000000 0.000000000 0.000000000
##   4 0.000000000 0.000000000 0.000000000 0.010752688 0.000000000 0.021505376
##   5 0.021276596 0.000000000 0.000000000 0.000000000 0.000000000 0.021276596
##    User country
## Y          Iran     Ireland      Israel       Italy       Japan      Jordan
##   1 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   2 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   3 0.000000000 0.022222222 0.000000000 0.000000000 0.022222222 0.022222222
##   4 0.010752688 0.021505376 0.000000000 0.010752688 0.000000000 0.000000000
##   5 0.000000000 0.028368794 0.014184397 0.000000000 0.000000000 0.000000000
##    User country
## Y         Kenya       Korea      Kuwait    Malaysia      Mexico Netherlands
##   1 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
```

```
##   2 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   3 0.000000000 0.000000000 0.000000000 0.000000000 0.022222222 0.022222222
##   4 0.000000000 0.010752688 0.000000000 0.010752688 0.010752688 0.010752688
##   5 0.000000000 0.000000000 0.000000000 0.007092199 0.028368794 0.014184397
##    User country
## Y    New Zeland      Norway Phillippines Puerto Rico Saudi Arabia    Scotland
##   1 0.166666667 0.000000000  0.000000000 0.000000000  0.000000000 0.000000000
##   2 0.000000000 0.000000000  0.000000000 0.000000000  0.000000000 0.000000000
##   3 0.022222222 0.000000000  0.022222222 0.000000000  0.000000000 0.000000000
##   4 0.021505376 0.000000000  0.000000000 0.000000000  0.000000000 0.000000000
##   5 0.007092199 0.000000000  0.000000000 0.007092199  0.000000000 0.000000000
##    User country
## Y     Singapore South Africa       Spain       Swiss Switzerland       Syria
##   1 0.000000000  0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   2 0.000000000  0.062500000 0.000000000 0.000000000 0.000000000 0.000000000
##   3 0.000000000  0.000000000 0.000000000 0.000000000 0.000000000 0.022222222
##   4 0.010752688  0.000000000 0.010752688 0.000000000 0.000000000 0.000000000
##   5 0.000000000  0.000000000 0.000000000 0.000000000 0.007092199 0.000000000
##    User country
## Y        Taiwan    Thailand          UK United Arab Emirates         USA
##   1 0.000000000 0.000000000 0.000000000          0.000000000 0.500000000
##   2 0.000000000 0.000000000 0.125000000          0.000000000 0.562500000
##   3 0.000000000 0.022222222 0.177777778          0.022222222 0.333333333
##   4 0.000000000 0.000000000 0.107526882          0.000000000 0.483870968
##   5 0.007092199 0.007092199 0.113475177          0.000000000 0.460992908
##
##    Period of stay
## Y     Dec-Feb   Jun-Aug   Mar-May   Sep-Nov
##   1 0.0000000 0.3333333 0.1666667 0.5000000
##   2 0.3125000 0.1250000 0.2500000 0.3125000
##   3 0.2222222 0.2000000 0.3333333 0.2444444
##   4 0.2365591 0.3655914 0.2473118 0.1505376
##   5 0.2340426 0.2553191 0.2269504 0.2836879
##
##    Traveler type
## Y     Business    Couples   Families    Friends       Solo
##   1 0.33333333 0.33333333 0.16666667 0.00000000 0.16666667
##   2 0.18750000 0.56250000 0.18750000 0.06250000 0.00000000
##   3 0.17777778 0.31111111 0.28888889 0.17777778 0.04444444
##   4 0.17204301 0.38709677 0.17204301 0.17204301 0.09677419
##   5 0.12765957 0.48936170 0.19148936 0.15602837 0.03546099
##
##    Pool
## Y            NO         YES
##   1 0.166666667 0.833333333
##   2 0.187500000 0.812500000
```

```
##   3 0.088888889 0.911111111
##   4 0.010752688 0.989247312
##   5 0.007092199 0.992907801
##
##    Gym
## Y           NO        YES
##   1 0.00000000 1.00000000
##   2 0.06250000 0.93750000
##   3 0.00000000 1.00000000
##   4 0.04301075 0.95698925
##   5 0.05673759 0.94326241
##
##    Tennis court
## Y          NO       YES
##   1 1.0000000 0.0000000
##   2 0.9375000 0.0625000
##   3 0.7777778 0.2222222
##   4 0.7311828 0.2688172
##   5 0.7446809 0.2553191
##
##    Spa
## Y          NO       YES
##   1 0.1666667 0.8333333
##   2 0.3125000 0.6875000
##   3 0.2444444 0.7555556
##   4 0.1935484 0.8064516
##   5 0.1914894 0.8085106
##
##    Casino
## Y           NO        YES
##   1 0.00000000 1.00000000
##   2 0.06250000 0.93750000
##   3 0.13333333 0.86666667
##   4 0.11827957 0.88172043
##   5 0.07092199 0.92907801
##
##    Free internet
## Y            NO         YES
##   1 0.166666667 0.833333333
##   2 0.187500000 0.812500000
##   3 0.111111111 0.888888889
##   4 0.032258065 0.967741935
##   5 0.007092199 0.992907801
##
##    Hotel stars
## Y            3          4          5         35         45
```

```
##   1 0.16666667 0.16666667 0.50000000 0.16666667 0.00000000
##   2 0.31250000 0.31250000 0.18750000 0.12500000 0.06250000
##   3 0.40000000 0.37777778 0.15555556 0.06666667 0.00000000
##   4 0.17204301 0.33333333 0.31182796 0.13978495 0.04301075
##   5 0.12765957 0.14184397 0.51773050 0.15602837 0.05673759
##
##    Nr. rooms
## Y          188         315         716         732         787         826
##   1 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   2 0.062500000 0.062500000 0.000000000 0.000000000 0.000000000 0.000000000
##   3 0.000000000 0.088888889 0.066666667 0.022222222 0.044444444 0.111111111
##   4 0.043010753 0.064516129 0.032258065 0.021505376 0.053763441 0.086021505
##   5 0.056737589 0.028368794 0.070921986 0.056737589 0.042553191 0.028368794
##    Nr. rooms
## Y         1228        1282        1467        2034        2700        2884
##   1 0.166666667 0.166666667 0.000000000 0.000000000 0.000000000 0.000000000
##   2 0.125000000 0.000000000 0.000000000 0.000000000 0.062500000 0.000000000
##   3 0.000000000 0.022222222 0.044444444 0.022222222 0.000000000 0.066666667
##   4 0.064516129 0.043010753 0.064516129 0.032258065 0.032258065 0.086021505
##   5 0.056737589 0.056737589 0.049645390 0.092198582 0.078014184 0.014184397
##    Nr. rooms
## Y         2916        2959        3003        3025        3348        3773
##   1 0.000000000 0.166666667 0.166666667 0.000000000 0.166666667 0.166666667
##   2 0.125000000 0.125000000 0.187500000 0.000000000 0.000000000 0.187500000
##   3 0.044444444 0.000000000 0.111111111 0.022222222 0.044444444 0.088888889
##   4 0.064516129 0.000000000 0.032258065 0.043010753 0.064516129 0.010752688
##   5 0.042553191 0.070921986 0.007092199 0.049645390 0.042553191 0.007092199
##    Nr. rooms
## Y         3933        3981        4027
##   1 0.000000000 0.000000000 0.000000000
##   2 0.000000000 0.062500000 0.000000000
##   3 0.022222222 0.155555556 0.022222222
##   4 0.053763441 0.064516129 0.043010753
##   5 0.056737589 0.021276596 0.070921986
##
##    User continent
## Y       Africa       Asia     Europe North America    Oceania South America
##   1 0.16666667 0.16666667 0.00000000    0.50000000 0.16666667    0.00000000
##   2 0.06250000 0.00000000 0.12500000    0.81250000 0.00000000    0.00000000
##   3 0.00000000 0.11111111 0.31111111    0.51111111 0.06666667    0.00000000
##   4 0.01075269 0.06451613 0.18279570    0.60215054 0.11827957    0.02150538
##   5 0.00000000 0.05673759 0.19858156    0.63120567 0.08510638    0.02836879
##
##    Member years
## Y       [,1]     [,2]
##   1 2.833333 3.250641
```

```
##   2 4.437500 3.182635
##   3 4.422222 2.444743
##   4 4.752688 3.041938
##   5 4.432624 2.957569
```

The output contains the apriori probabilities of the Score values, then the conditional probabilities of each class as a function of predictor values. The algorithm computes the probability that the score will be either of the 5 unique score values.

```
summary(lA_nb)
```

```
##           Length Class  Mode
## apriori    5     table  numeric
## tables    13     -none- list
## levels     5     -none- character
## isnumeric 13     -none- logical
## call       4     -none- call
```

Performance Evaluation

The model has an overall accuracy of 60.47% for the training data set.

Training

```
#Perfomance evaluation
library(caret)
# training
pred_class <- predict(lA_nb, newdata = train_LA)
confusionMatrix(pred_class, train_LA$Score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4   5
##          1   3   0   1   1   1
##          2   0   7   4   0   5
##          3   0   2  20   8  10
##          4   1   1   9  47  20
##          5   2   6  11  37 105
##
## Overall Statistics
##
##                Accuracy : 0.6047
##                  95% CI : (0.5469, 0.6603)
##     No Information Rate : 0.4684
##     P-Value [Acc > NIR] : 1.435e-06
##
##                   Kappa : 0.3883
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity          0.500000  0.43750  0.44444   0.5054   0.7447
## Specificity          0.989831  0.96842  0.92188   0.8510   0.6500
## Pos Pred Value       0.500000  0.43750  0.50000   0.6026   0.6522
## Neg Pred Value       0.989831  0.96842  0.90421   0.7937   0.7429
## Prevalence           0.019934  0.05316  0.14950   0.3090   0.4684
## Detection Rate       0.009967  0.02326  0.06645   0.1561   0.3488
## Detection Prevalence 0.019934  0.05316  0.13289   0.2591   0.5349
## Balanced Accuracy    0.744915  0.70296  0.68316   0.6782   0.6973
```

Validation

The model has 40.59% accuracy for the validation data.

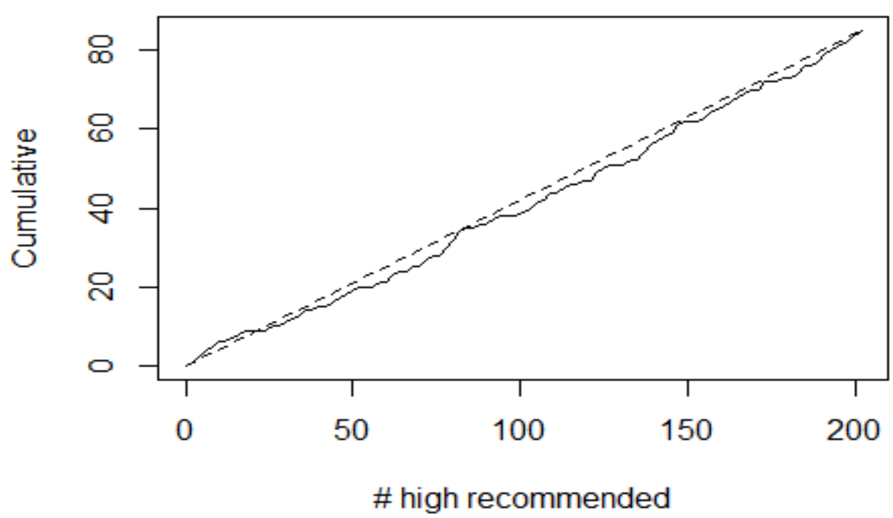
```
# validation
pred_class <- predict(lA_nb, newdata = valid_LA)
confusionMatrix(pred_class, valid_LA$Score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1  0  0  0  0  1
##          2  1  1  3  6  2
##          3  0  3  4 13  8
##          4  0  4  8 18 15
##          5  4  6 12 34 59
##
## Overall Statistics
##
##                Accuracy : 0.4059
##                  95% CI : (0.3376, 0.4771)
##     No Information Rate : 0.4208
##     P-Value [Acc > NIR] : 0.69
##
##                   Kappa : 0.0986
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.00000  0.07143   0.1481  0.25352   0.6941
## Specificity           0.99492  0.93617   0.8629  0.79389   0.5214
```

```
## Pos Pred Value        0.00000  0.07692   0.1429  0.40000   0.5130
## Neg Pred Value        0.97512  0.93122   0.8678  0.66242   0.7011
## Prevalence            0.02475  0.06931   0.1337  0.35149   0.4208
## Detection Rate        0.00000  0.00495   0.0198  0.08911   0.2921
## Detection Prevalence  0.00495  0.06436   0.1386  0.22277   0.5693
## Balanced Accuracy     0.49746  0.50380   0.5055  0.52371   0.6077
```

The lift chart is in support of the inadequacy of the model to fully classify the data.

```r
# predict probabilities
pred_prob <- predict(lA_nb, newdata = valid_LA, type = "raw")
#Lift Chart
library(gains)
gain <- gains(ifelse(valid_LA$Score == "5",1,0), pred_prob[,1], groups=100)
plot(c(0,gain$cume.pct.of.total*sum(valid_LA$Score=="5"))~c(0,gain$cume.obs),
     xlab="# high recommended", ylab="Cumulative", main="", type="l")
lines(c(0,sum(valid_LA$Score=="5"))~c(0, dim(valid_LA)[1]), lty=2)
```



Naive Bayes Lift Tree

Analysis of Results

The model has an overall accuracy of 60.47% for the training data set. The model has 40.59% accuracy for the validation data. The lift chart is in support of the inadequacy of the model to fully classify the data by being further away from the top-left upper corner.

Random forest

Implementation and Interpretation

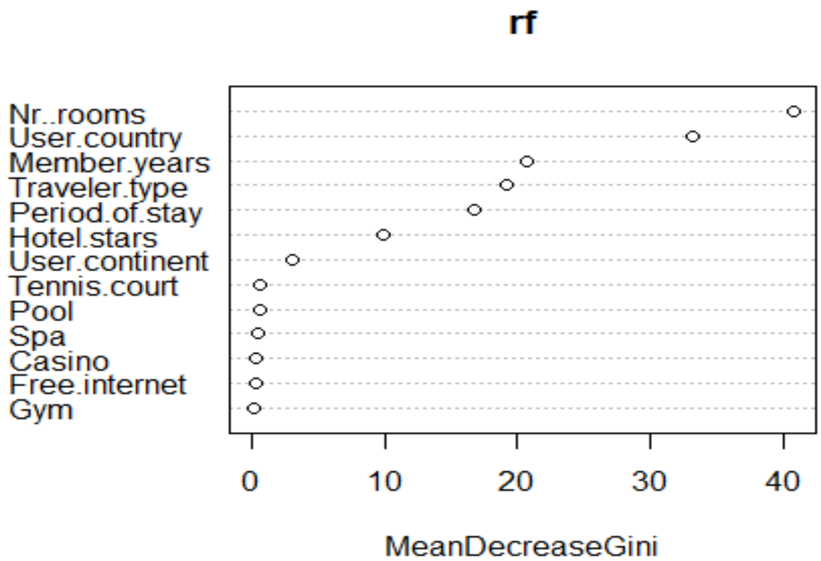
```
#partition
set.seed(1)
names(lasVegas) <- make.names(names(lasVegas))
train_index <- sample(c(1:dim(lasVegas)[1]), dim(lasVegas)[1]*0.6)
train_la <- lasVegas[train_index, ]
valid_la <- lasVegas[-train_index, ]
```

In the implementation, all 13 predictors are considered for each tree split.

```
library(randomForest)
## random forest
rf <- randomForest(Score ~ . , data = train_la, ntree = 500,
                   mtry = 13, nodesize = 5, importance = TRUE)
rf
## Call:
##  randomForest(formula = Score ~ ., data = train_la, ntree = 500,      mtry = 13, nodesize = 5, i
mportance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##         OOB estimate of  error rate: 55.48%
## Confusion matrix:
##   1 2 3  4  5 class.error
## 1 0 0 1  3  2   1.0000000
## 2 0 3 2  4  7   0.8125000
## 3 0 3 7 19 16   0.8444444
## 4 0 1 8 28 56   0.6989247
## 5 0 1 5 39 96   0.3191489
```

Perfomance Evaluation

```
## variable importance plot
varImpPlot(rf, type = 2)
```



Variance Importance

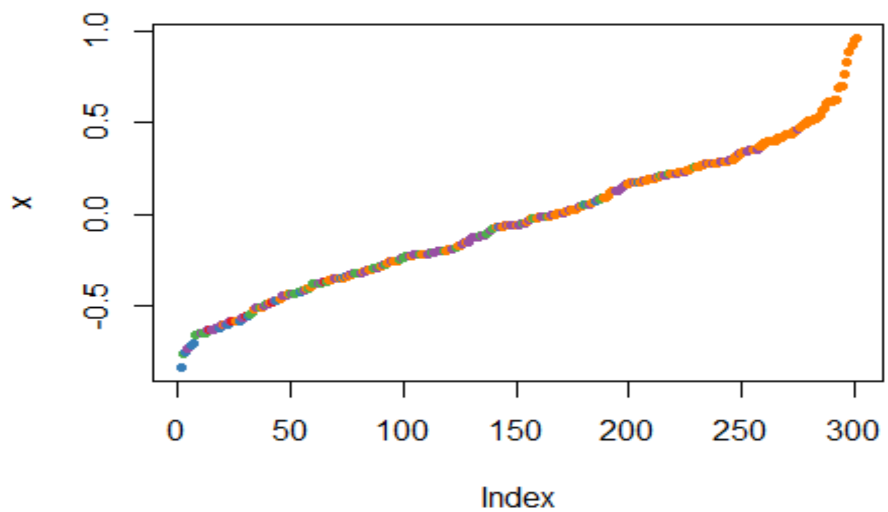
```
## confusion matrix
rf_pred <- predict(rf, valid_la)
confusionMatrix(rf_pred, valid_la$Score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1  0  0  0  0  0
##          2  0  0  0  2  0
##          3  1  1  4 14  3
##          4  1  5 12 22 27
##          5  3  8 11 33 55
##
## Overall Statistics
##
##                Accuracy : 0.401
##                  95% CI : (0.3328, 0.4721)
##     No Information Rate : 0.4208
##     P-Value [Acc > NIR] : 0.7386
##
##                   Kappa : 0.0617
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

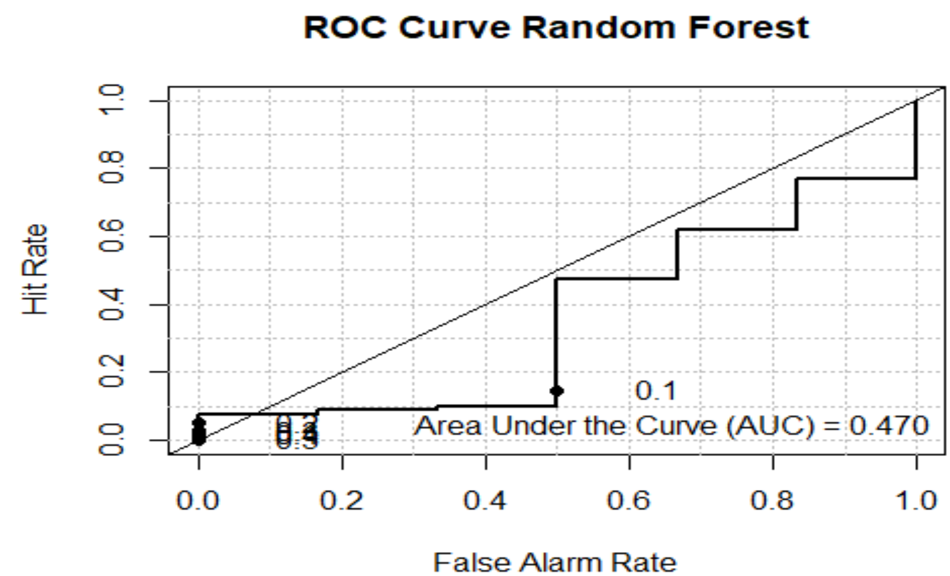
```
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.00000 0.000000   0.1481   0.3099   0.6471
## Specificity           1.00000 0.989362   0.8914   0.6565   0.5299
## Pos Pred Value            NaN 0.000000   0.1739   0.3284   0.5000
## Neg Pred Value        0.97525 0.930000   0.8715   0.6370   0.6739
## Prevalence            0.02475 0.069307   0.1337   0.3515   0.4208
## Detection Rate        0.00000 0.000000   0.0198   0.1089   0.2723
## Detection Prevalence  0.00000 0.009901   0.1139   0.3317   0.5446
## Balanced Accuracy     0.50000 0.494681   0.5198   0.4832   0.5885
```

The margin is evenly distributed across the negative and positive side and we cannot completely state that there was correct classification.

```
plot(margin(rf, testData$Species))
```




```
# Plot the OOB ROC curve.

library(verification)
aucc <- verification::roc.area(as.integer(as.factor(train_la$Score))-1,
              rf$votes[,2])$A
verification::roc.plot(as.integer(as.factor(train_la$Score))-1,
       rf$votes[,2], main="")
legend("bottomright", bty="n",
      sprintf("Area Under the Curve (AUC) = %1.3f", aucc))
title(main="ROC Curve Random Forest")
```



random forest lift

Analysis of Results

The model has an accuracy of 39.11% on the validation data. From the variable importance, we can see that the number of rooms has the highest score, whereas the gym has the lowest. However, the lift curve that shifts away from the top left corner show the poor performance of the model.

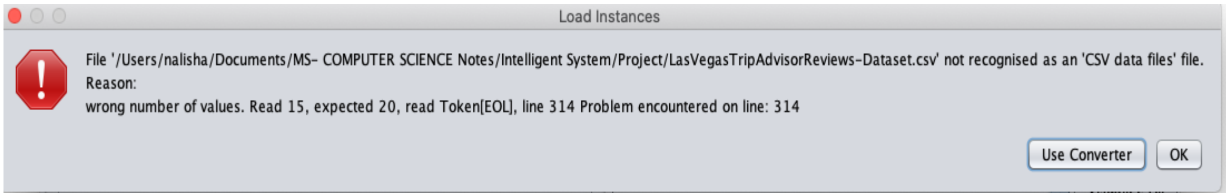
Conclusion

Based on the accuracy, the naive bayes with 60.47 and the random forest at 40.1, we can use the naive bayes to provide better prediction of the Score, and therefore able to select and recommend a better hotel based on the attributes required. However, the performance of both models is very low and may require adjustments.

The provision of a more and detailed attributes along with more observations around the hotels can be able to provide a more accurate prediction for classification. The metrics for effectiveness of models were based on the results provided by accompanying functions of the algorithms.

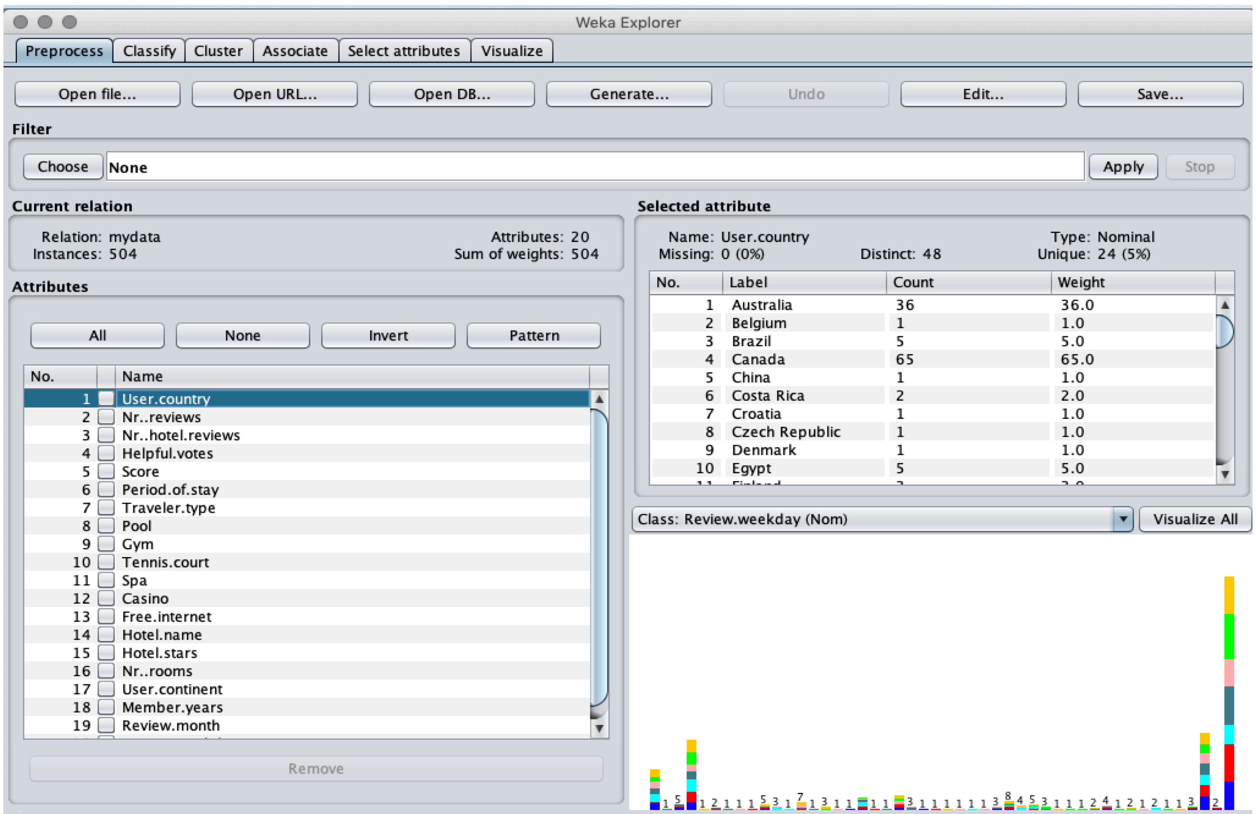
Data Preprocessing with WEKA

The very first step in WEKA was to upload the database, when I uploaded the database, I got this error



To solve this error in the convertor I set the 'fieldSeparator' as ';' and 'missingValue' as ' '

And then we converted it into .arrf file from R . Below is the .arff file visualization in weka



Cleaning the Data

Make numeric attribute nominal

In the data few of the values were numeric and nominal.

In WEKA, once you load the .arff file in preprocess tab

choose → Filter → unsupervised → attributes →NumericToNominal → Apply.

Handle outliers

Filter option in WEKA I used to detect any outlier

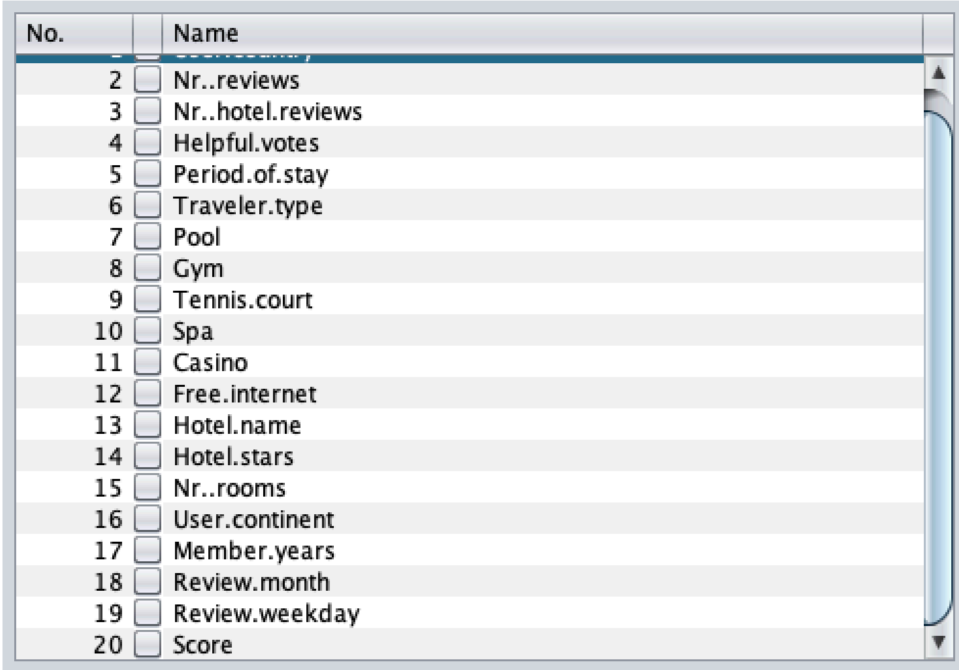
Preprocess Tab → Choose Filter → Filter → unsupervised → attributes → InterquartileRange → Apply.

We select score as the class Attribute.

Select the attribute as a class attribute we need to follow steps like

Preprocess Tab → Edit → Right click on Score attribute → Select 'Atrribute as class'

After selecting score as a class attribute, it moved down at the bottom in the list



No.		Name
2		Nr..reviews
3		Nr..hotel.reviews
4		Helpful.votes
5		Period.of.stay
6		Traveler.type
7		Pool
8		Gym
9		Tennis.court
10		Spa
11		Casino
12		Free.internet
13		Hotel.name
14		Hotel.stars
15		Nr..rooms
16		User.continent
17		Member.years
18		Review.month
19		Review.weekday
20		Score

Data Visualization

Association Analysis

Association rules in our dataset by using Apriori Algorithm.

```
Apriori
=======

Minimum support: 0.85 (428 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 3

Generated sets of large itemsets:

Size of set of large itemsets L(1): 4

Size of set of large itemsets L(2): 6

Size of set of large itemsets L(3): 1

Best rules found:

 1. Gym=YES 480 ==> Pool=YES 456    <conf:(0.95)> lift:(1) lev:(-0) [-1] conv:(0.91)
 2. Pool=YES 480 ==> Gym=YES 456    <conf:(0.95)> lift:(1) lev:(-0) [-1] conv:(0.91)
 3. Free.internet=YES 480 ==> Pool=YES 456    <conf:(0.95)> lift:(1) lev:(-0) [-1] conv:(0.91)
 4. Pool=YES 480 ==> Free.internet=YES 456    <conf:(0.95)> lift:(1) lev:(-0) [-1] conv:(0.91)
 5. Free.internet=YES 480 ==> Gym=YES 456    <conf:(0.95)> lift:(1) lev:(-0) [-1] conv:(0.91)
 6. Gym=YES 480 ==> Free.internet=YES 456    <conf:(0.95)> lift:(1) lev:(-0) [-1] conv:(0.91)
 7. Casino=YES 456 ==> Pool=YES 432    <conf:(0.95)> lift:(0.99) lev:(-0) [-2] conv:(0.87)
 8. Casino=YES 456 ==> Gym=YES 432    <conf:(0.95)> lift:(0.99) lev:(-0) [-2] conv:(0.87)
 9. Casino=YES 456 ==> Free.internet=YES 432    <conf:(0.95)> lift:(0.99) lev:(-0) [-2] conv:(0.87)
10. Gym=YES Free.internet=YES 456 ==> Pool=YES 432    <conf:(0.95)> lift:(0.99) lev:(-0) [-2] conv:(0.87)
```

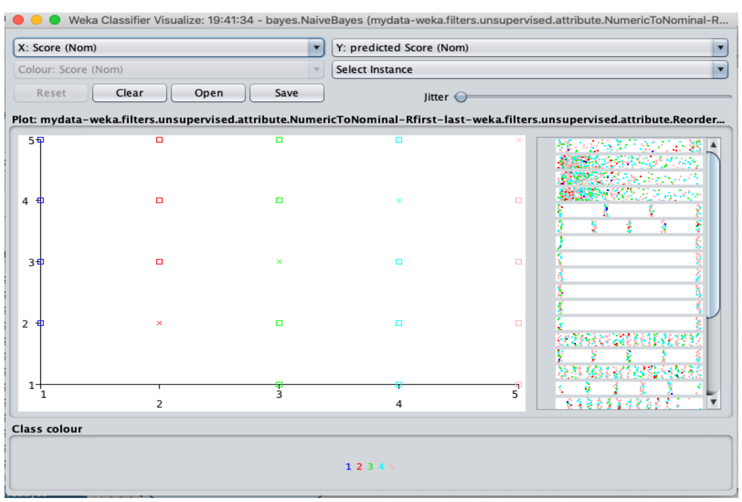
Apriori algorithm always generate best rule out of our dataset. We got the rules mainly on the attributes which have values Yes or No. Example Hotels which has Gym mostly has a Pool, Hotels which has free internet mostly has a pool.

Classification

1. Naïve bayes

```
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         218               43.254  %
Incorrectly Classified Instances       286               56.746  %
Kappa statistic                          0.1192
Mean absolute error                      0.2396
Root mean squared error                  0.4058
Relative absolute error                 89.614  %
Root relative squared error            111.1048 %
Total Number of Instances              504

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.012    0.000      0.000    0.000      -0.016   0.370     0.018     1
                 0.133    0.036    0.190      0.133    0.157      0.115    0.604     0.095     2
                 0.139    0.125    0.156      0.139    0.147      0.015    0.630     0.185     3
                 0.317    0.253    0.377      0.317    0.344      0.067    0.551     0.348     4
                 0.670    0.444    0.553      0.670    0.606      0.225    0.649     0.572     5
Weighted Avg.    0.433    0.303    0.405      0.433    0.415      0.132    0.606     0.403

=== Confusion Matrix ===

   a   b   c   d   e   <-- classified as
   0   1   1   1   8 |   a = 1
   0   4   3   9  14 |   b = 2
   2   5  10  31  24 |   c = 3
   1   6  28  52  77 |   d = 4
   3   5  22  45 152 |   e = 5
```

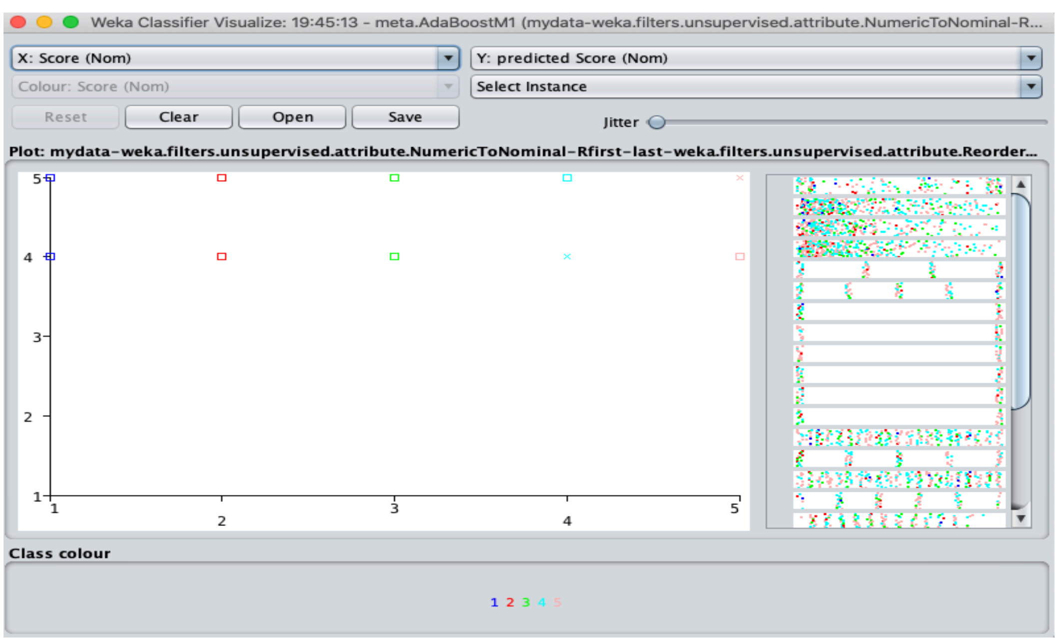
Visualization of classifier



Boosting:

```
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         226               44.8413 %
Incorrectly Classified Instances       278               55.1587 %
Kappa statistic                          0.1108
Mean absolute error                      0.2583
Root mean squared error                  0.3596
Relative absolute error                 96.6069 %
Root relative squared error             98.4663 %
Total Number of Instances              504

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.000    0.000    ?          0.000    ?          ?        0.393     0.019     1
                 0.000    0.000    ?          0.000    ?          ?        0.480     0.056     2
                 0.000    0.000    ?          0.000    ?          ?        0.575     0.164     3
                 0.628    0.535    0.361      0.628    0.459      0.088    0.528     0.336     4
                 0.542    0.347    0.562      0.542    0.552      0.196    0.591     0.523     5
Weighted Avg.    0.448    0.330    ?          0.448    ?          ?        0.557     0.372

=== Confusion Matrix ===

   a   b   c   d   e   <-- classified as
   0   0   0   5   6 |   a = 1
   0   0   0  19  11 |   b = 2
   0   0   0  54  18 |   c = 3
   0   0   0 103  61 |   d = 4
   0   0   0 104 123 |   e = 5
```

Visualization of classifier error

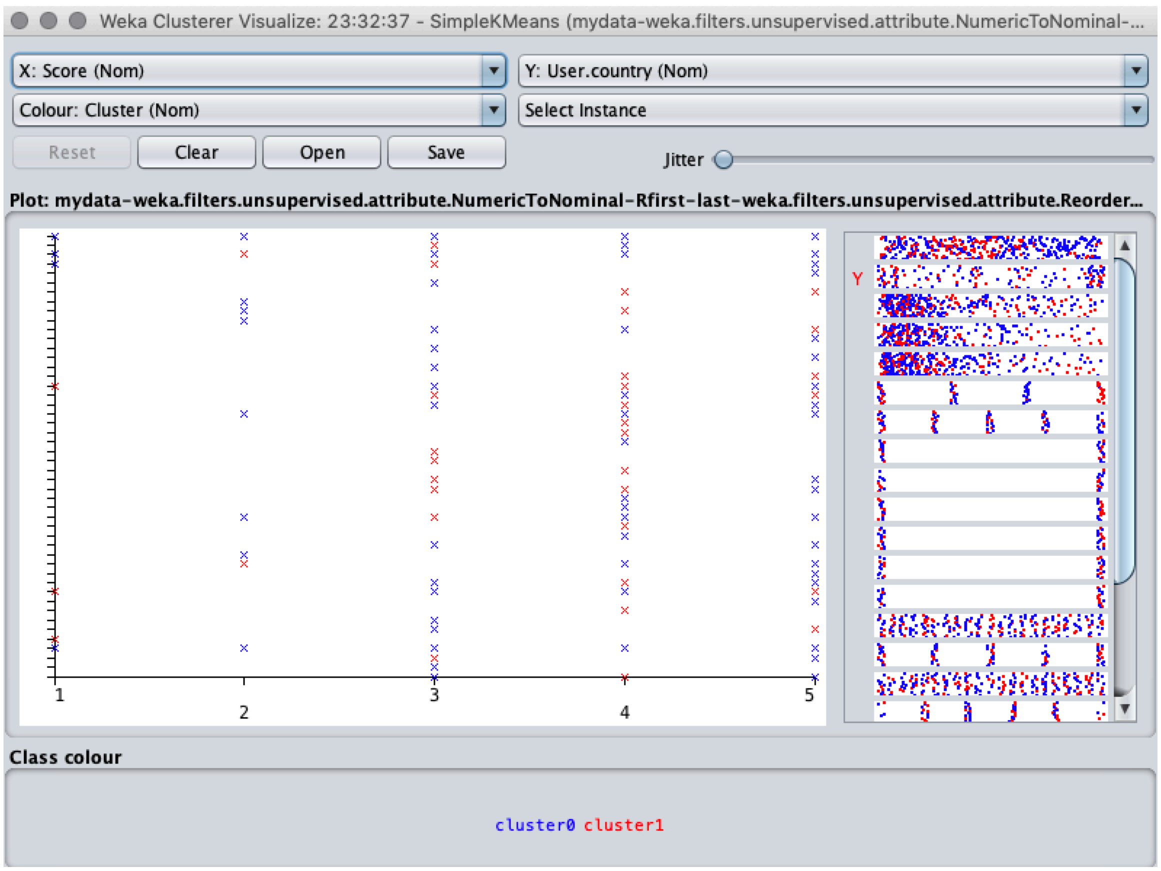


Clustering

Simple K Means clustering technique -

=== Model and evaluation on training set ===
Clustered Instances

0   330 (65%)
1   174 (35%)
The visualized cluster assignment where the Score is on X axis looks like



Attribute Selection

1. **Information Gain**
   Maximum gain is from 'Number of reviews' attribute. This attribute is most important attribute because traveler will decide hotel on the basis of number of reviews.

```
Attribute Evaluator (supervised, Class (nominal): 20 Score):
        Information Gain Ranking Filter

Ranked attributes:
 0.60166    2 Nr..reviews
 0.48422    4 Helpful.votes
 0.31221    3 Nr..hotel.reviews
 0.22486    1 User.country
 0.22482   15 Nr..rooms
 0.22482   13 Hotel.name
 0.09821   14 Hotel.stars
 0.0847    17 Member.years
 0.08086   18 Review.month
 0.0345    16 User.continent
 0.0338    19 Review.weekday
 0.0292     7 Pool
 0.02854    6 Traveler.type
 0.0282    12 Free.internet
 0.01511    5 Period.of.stay
 0.00844    9 Tennis.court
 0.00462   11 Casino
 0.0021    10 Spa
 0.00178    8 Gym
 0         21 Outlier
 0         22 ExtremeValue

Selected attributes: 2,4,3,1,15,13,14,17,18,16,19,7,6,12,5,9,11,10,8,21,22 : 21
```

Conclusion

Different data mining techniques used on the LasVegas strip dataset. We implement pre-processing filters like numericToNominal, selecting class label. Also handle the attribute which has has multiple values in a columns(hotel.star). Applied association analysis with Apriori algorithm. Applied classification techniques like Naïve Bayes and Boosting, also performed Clustering, Attribute selection on dataset.