STA380_Homework2_Sharma,Kartik

Kartik Sharma

August 18, 2015

Question 1

The task is to analyze the Austin Bergstorm International Airport flights data and come up with insights about the flights travelling from/to Austin during year 2008.

```
library(ggplot2)
library(RColorBrewer)

# Reading the csv file
airline<-
read.csv("C:/Users/Soundgarden/Documents/GitHub/STA380/data/ABIA.csv")
attach(airline)
str(airline)
```

```
## 'data.frame':    99260 obs. of  29 variables:
##  $ Year             : int  2008 2008 2008 2008 2008 2008 2008 2008 2008
2008 ...
##  $ Month            : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ DayofMonth       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ DayOfWeek        : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ DepTime          : int  120 555 600 601 601 636 646 650 650 654 ...
##  $ CRSDepTime       : int  1935 600 600 605 600 645 655 700 650 700 ...
##  $ ArrTime          : int  309 826 728 727 654 934 735 841 1139 1117 ...
##  $ CRSArrTime       : int  2130 835 729 750 700 932 750 857 1145 1133 ...
##  $ UniqueCarrier    : Factor w/ 16 levels "9E","AA","B6",..: 1 2 16 1 2 9
4 15 2 3 ...
##  $ FlightNum        : int  5746 1614 2883 5743 1157 1674 340 541 1182 1060
...
##  $ TailNum          : Factor w/ 2626 levels "","80009E","80019E",..: 28
1095 2379 111 1272 2567 273 394 1255 483 ...
##  $ ActualElapsedTime: int  109 151 148 86 53 178 49 111 169 203 ...
##  $ CRSElapsedTime   : int  115 155 149 105 60 167 55 117 175 213 ...
##  $ AirTime          : int  88 133 125 70 38 145 28 94 153 177 ...
##  $ ArrDelay         : int  339 -9 -1 -23 -6 2 -15 -16 -6 -16 ...
##  $ DepDelay         : int  345 -5 0 -4 1 -9 -9 -10 0 -6 ...
##  $ Origin           : Factor w/ 53 levels "ABQ","ATL","AUS",..: 32 3 3 3 3
3 3 29 25 3 ...
##  $ Dest             : Factor w/ 53 levels "ABQ","ATL","AUS",..: 3 39 42 33
12 34 21 3 3 24 ...
##  $ Distance         : int  559 978 872 559 190 1042 140 650 1242 1522 ...
```

```
##  $ TaxiIn           : int  3 7 7 4 5 11 6 6 4 13 ...
##  $ TaxiOut          : int  18 11 16 12 10 22 15 11 12 13 ...
##  $ Cancelled        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ CancellationCode : Factor w/ 4 levels "","A","B","C": 1 1 1 1 1 1 1 1 1
1 ...
##  $ Diverted         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ CarrierDelay     : int  339 NA NA NA NA NA NA NA NA NA ...
##  $ WeatherDelay     : int  0 NA NA NA NA NA NA NA NA NA ...
##  $ NASDelay         : int  0 NA NA NA NA NA NA NA NA NA ...
##  $ SecurityDelay    : int  0 NA NA NA NA NA NA NA NA NA ...
##  $ LateAircraftDelay: int  0 NA NA NA NA NA NA NA NA NA ...
```

On analyzing the structure of the airline data we see that many variables have NA values for most of the data.

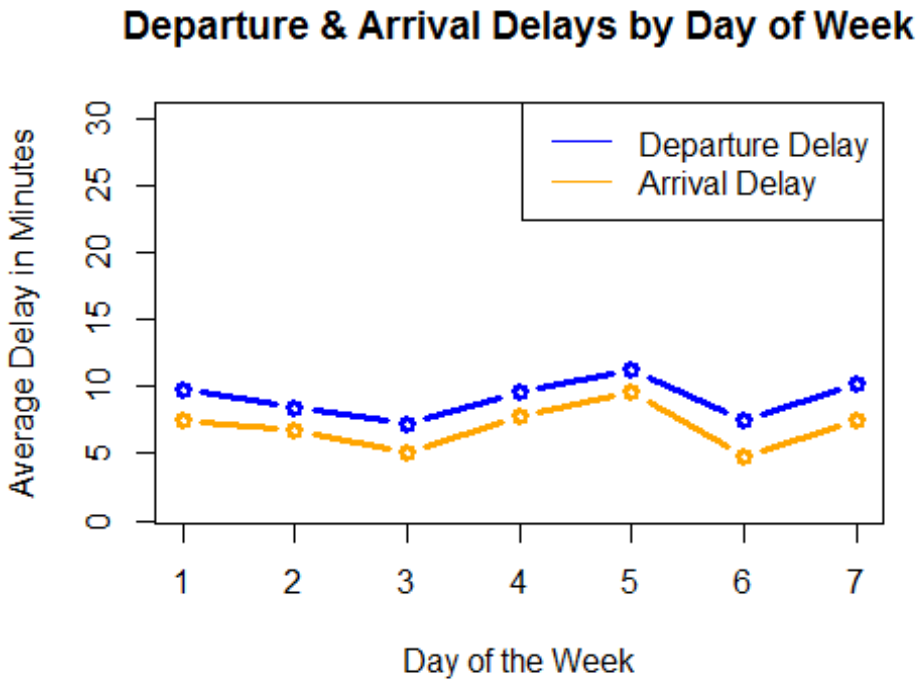
Replacing the NA values with zeroes

```
# Replacing all the na values in
CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay columns
with zero

vars.to.replace =
c("CarrierDelay","WeatherDelay","NASDelay","SecurityDelay","LateAircraftDelay
")
airline1 = airline[vars.to.replace]
airline1[is.na(airline1)]=0
airline[vars.to.replace]= airline1

airline$DepTime_hour=as.integer(DepTime/100)
airline$ArrTime_hour=as.integer(ArrTime/100)
airline$TotalDelay = abs(airline$ArrDelay) + abs(airline$DepDelay) +
abs(airline$CarrierDelay) + abs(airline$WeatherDelay)+ abs(airline$NASDelay)
+ abs(airline$SecurityDelay) + abs(airline$LateAircraftDelay)
```

Creating exploratory plot for delays for each day of the week. Since most of the delays due to the flight reasons are the arrival and the departure delays. Thus, we shall explore these two types only in the following plots.

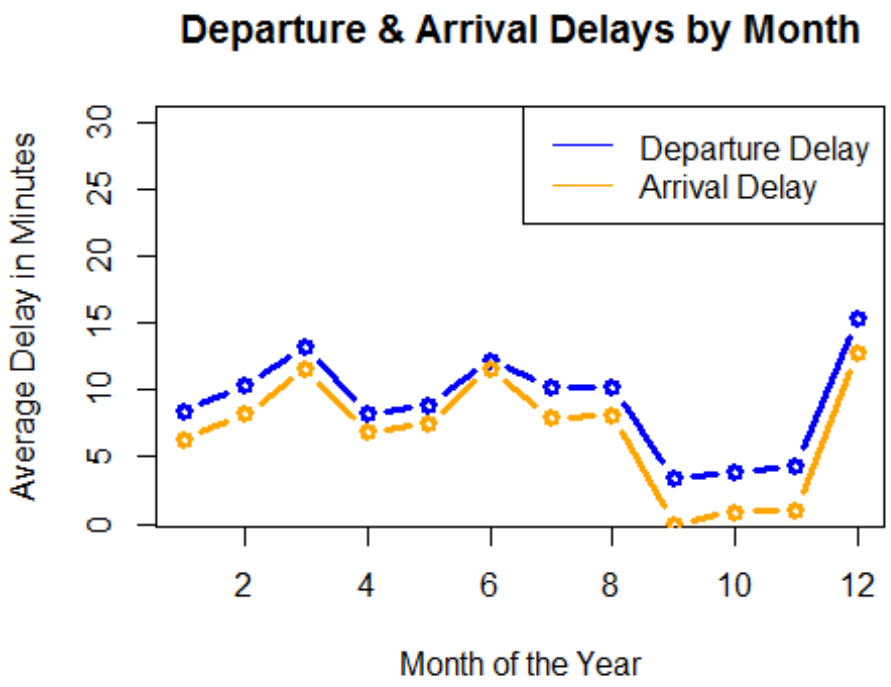
```
avgdepdelay = aggregate(DepDelay,by = list(DayOfWeek), FUN = mean, na.rm=
TRUE)
avgarrdelay = aggregate(ArrDelay,by = list(DayOfWeek), FUN = mean, na.rm=
TRUE)

#Plot the average arrival / departure delay time against the day of the week

plot(avgdepdelay$x, type ="b", xlab = "Day of the Week", ylab = "Average
Delay in Minutes", col = "Blue", lwd = 3, ylim = c(1,30), main = "Departure &
Arrival Delays by Day of Week" )
lines(avgarrdelay$x, type = "b", col = "Orange", lwd = 3)
```

```
legend ("topright", c("Departure Delay", "Arrival Delay"), lty = 1, col =
c('Blue','Orange'))
```



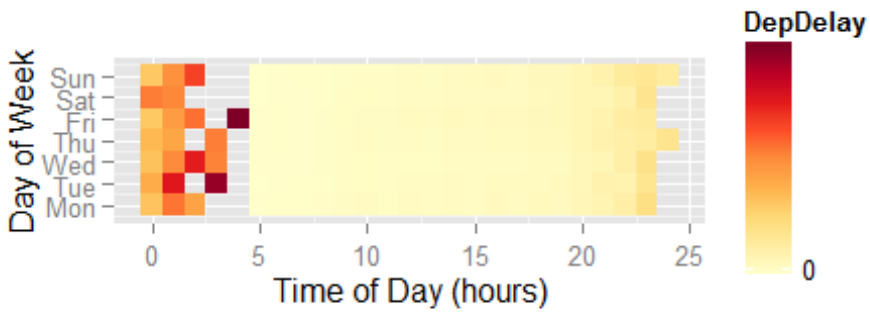
Creating exploratory plot for delays for each month of the year

```
#Aggregating the departure and arrival delays by month of the year
avg_depdelay_month = aggregate(DepDelay,by = list(Month), FUN = mean, na.rm=
TRUE)
avg_arrdelay_month = aggregate(ArrDelay,by = list(Month), FUN = mean, na.rm=
TRUE)

#Plot the average arrival / departure delay time against the month
plot(avg_depdelay_month$x, type ="b", xlab = "Month of the Year", ylab =
"Average Delay in Minutes", col = "Blue", lwd = 3, ylim = c(1,30), main =
"Departure & Arrival Delays by Month")
lines(avg_arrdelay_month$x, type = "b", col = "Orange", lwd = 3)
legend ("topright", c("Departure Delay", "Arrival Delay"), lty = 1, col =
c('Blue','Orange'))
```



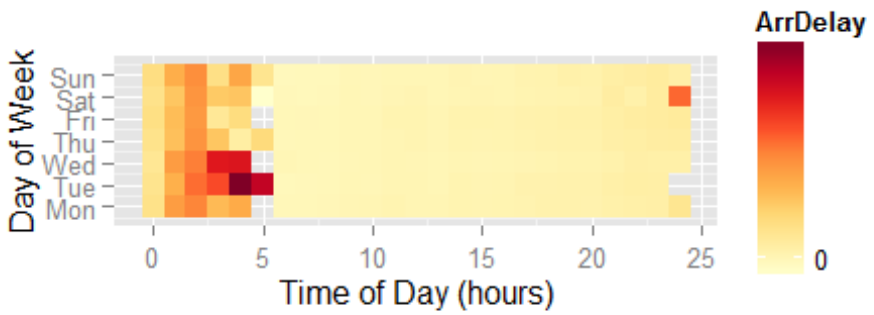
Creating heat maps for the departure delays for different time and days

```
departure_delays =
aggregate(DepDelay~DayOfWeek+DepTime_hour,airline,FUN='mean')

ggplot(departure_delays, aes(DepTime_hour,y=DayOfWeek))+
  geom_tile(aes(fill=DepDelay))+
  scale_fill_gradientn(colours=brewer.pal(9,"YlOrRd"),
                       breaks=seq(0,max(departure_delays$DepDelay),by=3000))+

scale_y_continuous(breaks=1:7,labels=c("Mon","Tue","Wed","Thu","Fri","Sat","S
un"))+
  labs(x="Time of Day (hours)", y="Day of Week")+ coord_fixed()
```



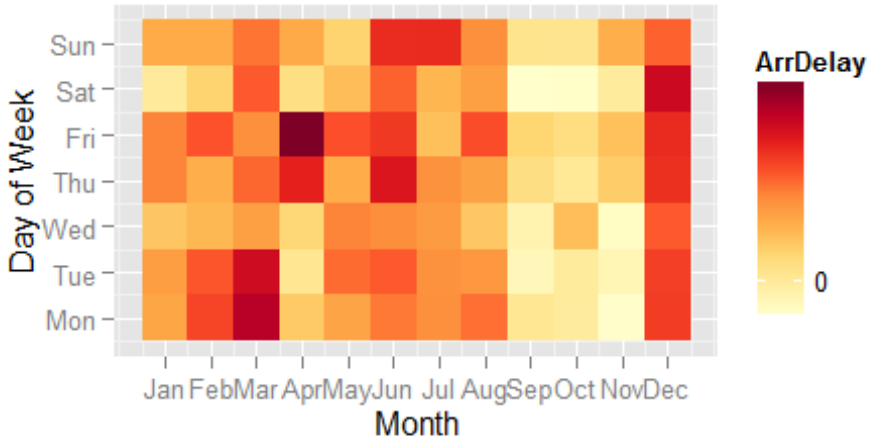
Creating heat maps for the arrival delays for different time and days

```
arrival_delays= aggregate(ArrDelay~DayOfWeek+ArrTime_hour,airline,FUN='mean')

ggplot(arrival_delays, aes(ArrTime_hour,y=DayOfWeek))+
  geom_tile(aes(fill=ArrDelay))+
  scale_fill_gradientn(colours=brewer.pal(9,"YlOrRd"),
                       breaks=seq(0,max(arrival_delays$ArrDelay),by=3000))+

scale_y_continuous(breaks=1:7,labels=c("Mon","Tue","Wed","Thu","Fri","Sat","S
un"))+
  labs(x="Time of Day (hours)", y="Day of Week")+ coord_fixed()
```



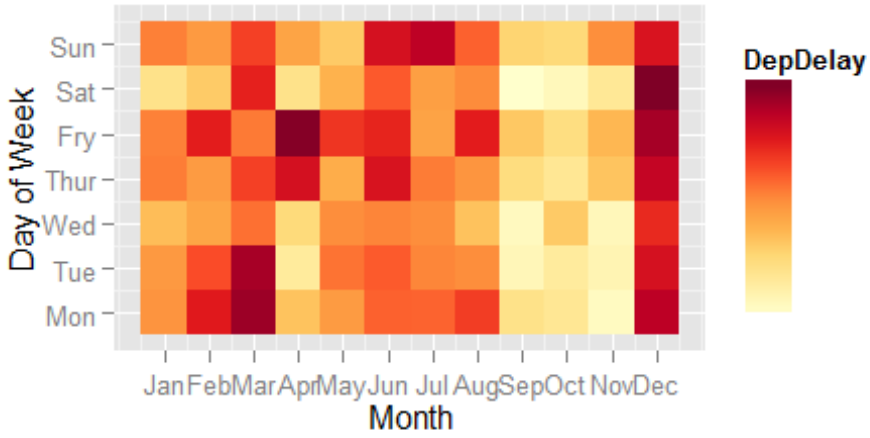
Creating heat maps for the arrival delays for different days of week and months

```
arr_agg_month= aggregate(ArrDelay~DayOfWeek+Month,airline,FUN='mean')

ggplot(arr_agg_month, aes(Month,y=DayOfWeek))+
  geom_tile(aes(fill=ArrDelay))+
  scale_fill_gradientn(colours=brewer.pal(9,"YlOrRd"),
                       breaks=seq(0,max(arr_agg_month$ArrDelay),by=3000))+

scale_y_continuous(breaks=1:7,labels=c("Mon","Tue","Wed","Thu","Fri","Sat","S
un"))+
  scale_x_continuous(breaks=1:12,
labels=c("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","D
ec"))+
  labs(x="Month", y="Day of Week")+ coord_fixed()
```



Creating heat maps for the departure delays for different days of week and months

```
Dep_agg_month    <- aggregate(DepDelay~DayOfWeek+Month,airline,FUN='mean')

ggplot(Dep_agg_month, aes(Month,y=DayOfWeek))+
  geom_tile(aes(fill=DepDelay))+
  scale_fill_gradientn(colours=brewer.pal(9,"YlOrRd"),
                       breaks=seq(0,max(Dep_agg_month$DepDelay),by=3000))+

scale_y_continuous(breaks=7:1,labels=c("Sun","Sat","Fry","Thur","Wed","Tue","
Mon"))+ scale_x_continuous(breaks=1:12,
labels=c("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","D
ec"))+
  labs(x="Month", y="Day of Week")+ coord_fixed()
```



Looking at the above plots we find that we have the highest average delay on Friday.Also, we have the highest average delay by month in Decemeber, June, and March.

This is an expected trend as these months coincide with the popular holiday seasons and we expect higher passenger volume during these periods.

Most of the delays occur in the wee hours specially in Fridays and Tuesdays.

Also, fridays have the longest delays on average with december being the worst.

Question 2

In order to predict the authors for the different files, we use Naive Bayes & Random Forest techniquees.

We start off by loading the libraries required for the analysis.

```
# Loading the necessary libraries

library(tm)

## Loading required package: NLP

library(randomForest)
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.

library(e1071)

## Warning: package 'e1071' was built under R version 3.2.2

library(rpart)
library(ggplot2)

##
## Attaching package: 'ggplot2'
##
## The following object is masked from 'package:NLP':
##
##     annotate

library(caret)

## Loading required package: lattice

library (plyr)
```

Creating the reader function

```
#reader function

readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)), id=fname, language='en') }
```

We make up a training corpus by reading in all the files for different authors in the training set.

We can use for loop to iterate over different files in the training set and reading them sequentially.

```
#TRAINING CORPUS
author_directory =
Sys.glob('C:/Users/Soundgarden/Documents/GitHub/STA380/data/ReutersC50/C50train/*')
file_list = NULL
train_labels = NULL
for(author in author_directory) {
  author_name = substring(author, first=74)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  train_labels = append(train_labels, rep(author_name, length(files_to_add)))
}
```

Cleaning up the read documents and initializing training corpus.

```
# Named conversion & cleanup
all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
```

```
names(all_docs) = sub('.txt', '', names(all_docs))

#Initialize Training Corpus
train_corpus = Corpus(VectorSource(all_docs))
names(train_corpus) = file_list
```

Cleaning up the output training corpus

```
#Tokenization of training Corpus

#Converting to lowercase
train_corpus = tm_map(train_corpus, content_transformer(tolower))

# Removing the numbers
train_corpus = tm_map(train_corpus, content_transformer(removeNumbers))

#Removing Punctuation Marks
train_corpus = tm_map(train_corpus, content_transformer(removePunctuation))

#Stripping white spaces
train_corpus = tm_map(train_corpus, content_transformer(stripWhitespace))

# Removing stop words
train_corpus = tm_map(train_corpus, content_transformer(removeWords),
stopwords("SMART"))
```

Creating a document term matrix for the training data

```
#Create training DTM & dense matrix
DTM_train = DocumentTermMatrix(train_corpus)
DTM_train = removeSparseTerms(DTM_train, 0.95)
```

Creating the testing data and cleaning up similar to the training data

```
#TESTING CORPUS

author_directory_test =
Sys.glob('C:/Users/Soundgarden/Documents/GitHub/STA380/data/ReutersC50/C50tes
t/*')
file_list = NULL
test_labels = NULL
for(author in author_directory_test) {
  author_name = substring(author, first=73)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  test_labels = append(test_labels, rep(author_name, length(files_to_add)))
}

# Named conversion & cleanup
all_docs = lapply(file_list, readerPlain)
```

```
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))

#Initialize Testing Corpus
test_corpus = Corpus(VectorSource(all_docs))
names(test_corpus) = file_list
```

Tokenization of Testing Corpus

```
#Converting to lowercase
test_corpus = tm_map(test_corpus, content_transformer(tolower))

# Removing the numbers
test_corpus = tm_map(test_corpus, content_transformer(removeNumbers))

#Removing Punctuation Marks
test_corpus = tm_map(test_corpus, content_transformer(removePunctuation))


#Stripping white spaces
test_corpus = tm_map(test_corpus, content_transformer(stripWhitespace))

# Removing stop words
test_corpus = tm_map(test_corpus, content_transformer(removeWords),
stopwords("SMART"))
```

Dictionary Creation: Creating a dictionary of all the terms occuring in the training corpus so that they can be

used to create a matrix for predicting the testing document terms

```
# We need a dictionary of terms from the training corpus in order to extract
terms from the test corpus
training_dictionary= NULL
training_dictionary= dimnames(DTM_train)[[2]]

#Create testing DTM & matrix using dictionary words only
DTM_test = DocumentTermMatrix(test_corpus, list(dictionary=
training_dictionary))
DTM_test = removeSparseTerms(DTM_test, 0.95)

#Convert DTMs into Data Frames for use in classifier models

DTM_training_df = as.data.frame(inspect(DTM_train))
DTM_testing_df = as.data.frame(inspect(DTM_test))
```

Applying Naive Bayes model to the training data to predict the outcomes for the test data.

We also use Laplacian smoothing for the Naive Bayes model

```
# Naive Bayes Model
model_NB = naiveBayes(x=DTM_training_df, y=as.factor(train_labels),
laplace=1)

# Predicting outcomes for the testing data
pred_NB = predict(model_NB, DTM_testing_df)

table_NB = as.data.frame(table(pred_NB,test_labels))

# Creating a confusion Matrix to check for the accuracy of the Naive Bayes
model.
conf_NB = confusionMatrix(table(pred_NB,test_labels))
```

Analyzing the results from the Naive Bayes Model

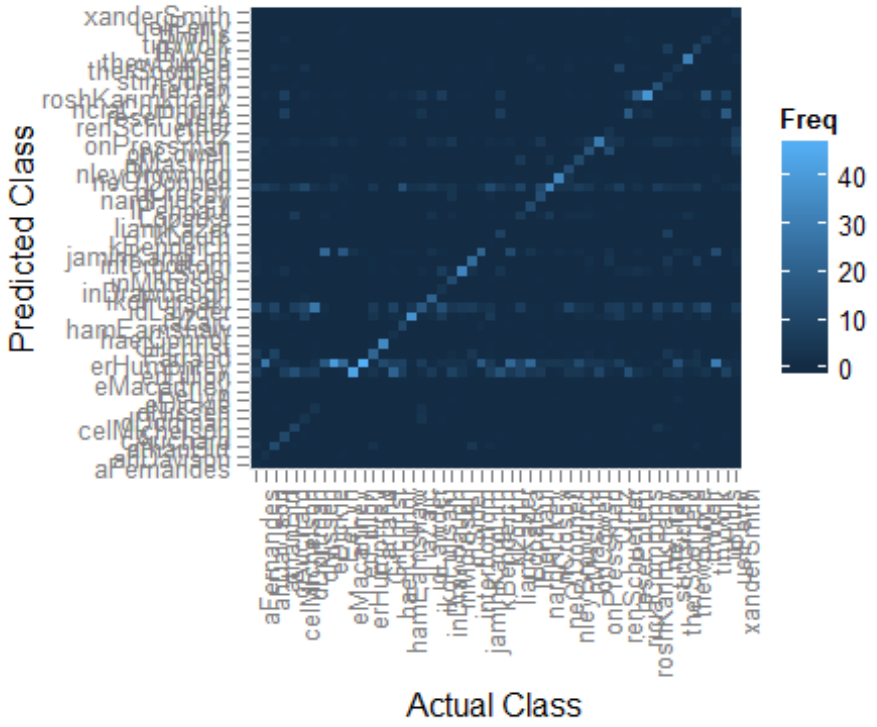
Looking at the following graph and the accuracy score of 74% we can say that Naive Bayes did a fairly decent

job of predicting the test data.

It signifies that approximately 74% times the model was able to correctly identify the correct author

on the test data set.

```
# Plotting the results for the Naive Bayes results
plot = ggplot(table_NB)
plot + geom_tile(aes(x=test_labels, y=pred_NB, fill=Freq)) +
  scale_x_discrete(name="Actual Class") +
  scale_y_discrete(name="Predicted Class") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




```
conf_NB$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.2796000      0.2648980      0.2620737      0.2976439      0.0200000
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Applying the random forest model to perform predictions.

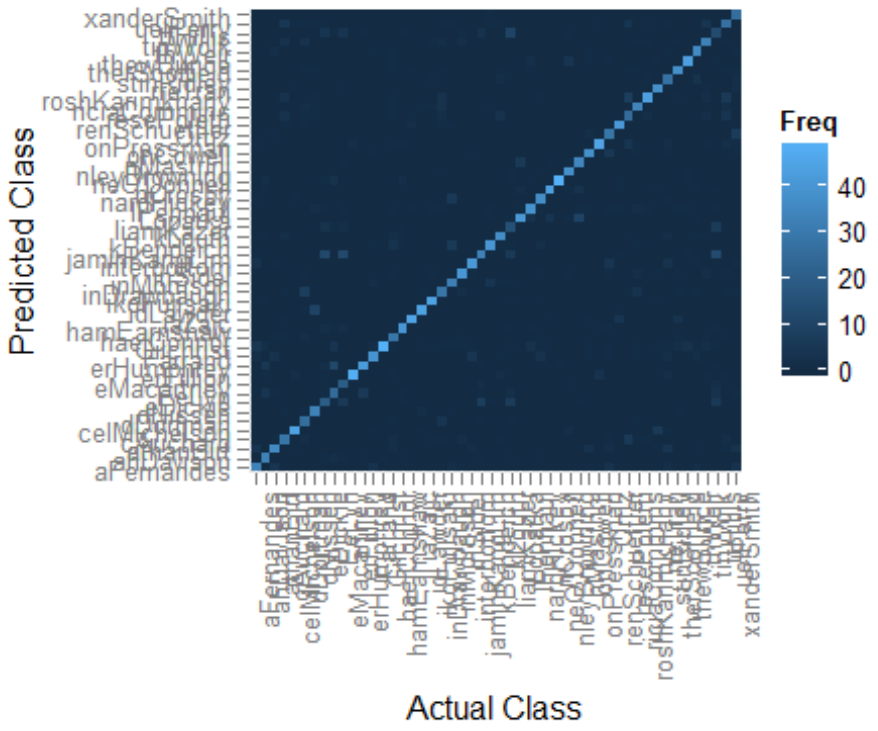
The difference between Naive Bayes and random forest for this analysis is that Naive bayes considers that

the words other than those in the training set will not occur. Random forest takes into account the non- occuring words. It requires the columns to be the same for both the training data and the test data.

Looking at the accuracy measures for random forest model, we're able to correctly identify the author in the test data with 69% accuracy.

```
#Random Forest Model

DTM_test = as.matrix(DTM_test)
DTM_train = as.matrix(DTM_train)

# Adding empty columns in the test data set for words that appear in
#the training data, but not in the test data.
#RandomForest requires the same variables in training and test sets
```

```
common <- data.frame(DTM_testing_df[,intersect(colnames(DTM_testing_df),
colnames(DTM_training_df))])
unknown <- read.table(textConnection(""), col.names =
colnames(DTM_training_df), colClasses = "integer")

RFM_test_clean = rbind.fill(common, unknown)

RFM_test_df = as.data.frame(RFM_test_clean)


randomforest_model = randomForest(x=DTM_training_df,
y=as.factor(train_labels), mtry=3, ntree=200)
pred_RF = predict(randomforest_model, data=RFM_test_clean)

table_RF = as.data.frame(table(pred_RF,test_labels))

plot = ggplot(table_RF)
plot + geom_tile(aes(x=test_labels, y=pred_RF, fill=Freq)) +
  scale_x_discrete(name="Actual Class") +
  scale_y_discrete(name="Predicted Class") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



```
confint_RF = confusionMatrix(table(pred_RF,test_labels))

confint_RF$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.7056000      0.6995918      0.6873006      0.7234166      0.0200000
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Overall, Naive Bayes did a better job of predicting the authors for the test dataset than the random forests

Question 3

```
# Loading the required libraries
library(arules)
```

```
## Warning: package 'arules' was built under R version 3.2.2
```

```
## Loading required package: Matrix
##
## Attaching package: 'arules'
##
## The following objects are masked from 'package:base':
##
##     %in%, write
```

```
# Read in groceries for the different transactions and creating a transaction object
# Since there can be multiple items in a single transaction we use the format as the basket for read. transactions command
groceries <-
read.transactions("C:/Users/Soundgarden/Documents/GitHub/STA380/data/groceries.txt", format = c("basket"), sep = ",", cols = NULL, rm.duplicates = TRUE, encoding = "unknown")

# Now run the 'apriori' algorithm on the transaction data to find the frequent items set
# Finding the rules with support > .01 & confidence >.5 & length (# artists) <= 5
grocery_cat <- apriori(groceries,
                       parameter=list(support=.01, confidence=.5, maxlen=5))
```

```
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.5    0.1    1 none FALSE            TRUE    0.01      1      5
##  target   ext
```

```
##   rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## apriori - find association rules with the apriori algorithm
## version 4.21 (2004.05.09)        (c) 1996-2004   Christian Borgelt
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

Inspecting the output of the apriori algorithm and creating meaningful subsets.

Looking at the output we find the 15 subsets in the data with high association

We looked at those subsets that had lift >3 / confidence>0.57 / support >0.015 and confidence>0.55

We used these subset parameters so as to come up with the subset that had relatively higher relevance and lift than the other subsets in the data

On analysisng the outputs we find that whole milk is bought together with curd & yogurt whereas people purchasing fruits and root vegetables had higher chances to purchase other vegetables as well.

```
library(arules)
# Look at the output
inspect(grocery_cat)
```

```
##    lhs                     rhs                    support confidence
lift
## 1  {curd,
##     yogurt}             => {whole milk}       0.01006609  0.5823529
2.279125
## 2  {butter,
##     other vegetables}   => {whole milk}       0.01148958  0.5736041
2.244885
## 3  {domestic eggs,
##     other vegetables}   => {whole milk}       0.01230300  0.5525114
2.162336
## 4  {whipped/sour cream,
##     yogurt}             => {whole milk}       0.01087951  0.5245098
2.052747
## 5  {other vegetables,
##     whipped/sour cream} => {whole milk}       0.01464159  0.5070423
```

```
1.984385
## 6  {other vegetables,
##     pip fruit}          => {whole milk}       0.01352313  0.5175097
2.025351
## 7  {citrus fruit,
##     root vegetables}    => {other vegetables} 0.01037112  0.5862069
3.029608
## 8  {root vegetables,
##     tropical fruit}     => {other vegetables} 0.01230300  0.5845411
3.020999
## 9  {root vegetables,
##     tropical fruit}     => {whole milk}       0.01199797  0.5700483
2.230969
## 10 {tropical fruit,
##     yogurt}             => {whole milk}       0.01514997  0.5173611
2.024770
## 11 {root vegetables,
##     yogurt}             => {other vegetables} 0.01291307  0.5000000
2.584078
## 12 {root vegetables,
##     yogurt}             => {whole milk}       0.01453991  0.5629921
2.203354
## 13 {rolls/buns,
##     root vegetables}    => {other vegetables} 0.01220132  0.5020921
2.594890
## 14 {rolls/buns,
##     root vegetables}    => {whole milk}       0.01270971  0.5230126
2.046888
## 15 {other vegetables,
##     yogurt}             => {whole milk}       0.02226741  0.5128806
2.007235

## Choose a subset
inspect(subset(grocery_cat, subset=lift > 3))

##   lhs                   rhs                   support confidence     lift
## 1 {citrus fruit,
##    root vegetables} => {other vegetables} 0.01037112  0.5862069 3.029608
## 2 {root vegetables,
##    tropical fruit}  => {other vegetables} 0.01230300  0.5845411 3.020999

inspect(subset(grocery_cat, subset=confidence > 0.58))

##   lhs                   rhs                   support confidence     lift
## 1 {curd,
##    yogurt}          => {whole milk}       0.01006609  0.5823529 2.279125
## 2 {citrus fruit,
##    root vegetables} => {other vegetables} 0.01037112  0.5862069 3.029608
## 3 {root vegetables,
##    tropical fruit}  => {other vegetables} 0.01230300  0.5845411 3.020999
```

```
inspect(subset(grocery_cat, subset=support > .015 & confidence > 0.55))
```

```
## NULL
```