Assignment 2

Sadavath Sharma

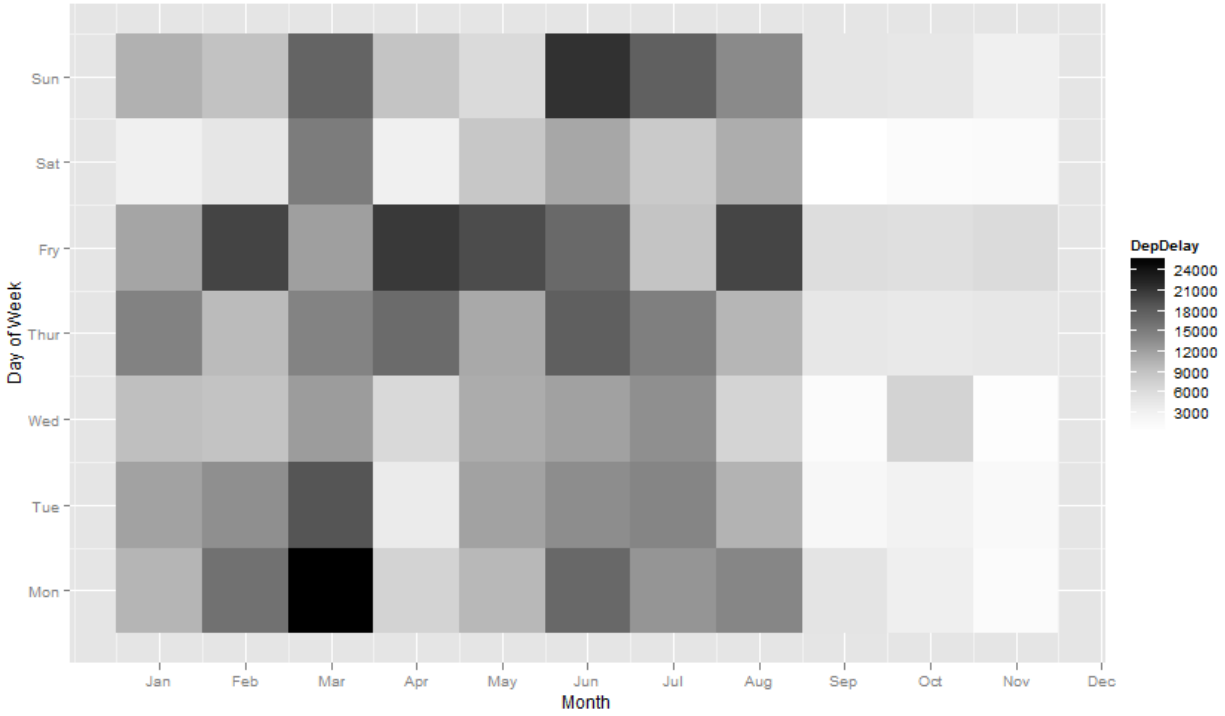
August 18, 2015

Question1:

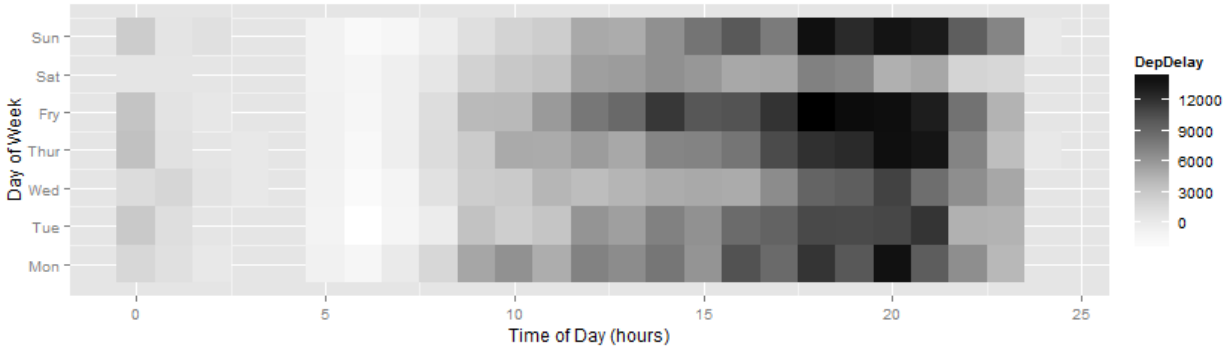
```
library(tm)

airports=
read.csv("https://raw.githubusercontent.com/jgscott/STA380/master/data/ABIA.csv")

airports$DepTime_hour=as.integer(DepTime/100)

airports$ArrTime_hour=as.integer(ArrTime/100)

attach(airports)

names(airports)

dep_agg    <- aggregate(DepDelay~DayOfWeek+DepTime_hour,airports,FUN='sum')

library(ggplot2)

library(RColorBrewer)

ggplot(dep_agg, aes(DepTime_hour,y=DayOfWeek))+

  geom_tile(aes(fill=DepDelay))+

  scale_fill_gradientn(colours=brewer.pal(9,"Greys"),

                       breaks=seq(0,max(dep_agg$DepDelay),by=3000))+

scale_y_continuous(breaks=7:1,labels=c("Sun","Sat","Fry","Thur","Wed","Tue","Mon"))+

  labs(x="Time of Day (hours)", y="Day of Week")+ coord_fixed()

arr_agg    <- aggregate(ArrDelay~DayOfWeek+ArrTime_hour,airports,FUN='sum')

ggplot(arr_agg, aes(ArrTime_hour,y=DayOfWeek))+

  geom_tile(aes(fill=ArrDelay))+

  scale_fill_gradientn(colours=brewer.pal(9,"Greys"),

                       breaks=seq(0,max(arr_agg$ArrDelay),by=3000))+
```

```
scale_y_continuous(breaks=7:1,labels=c("Sun","Sat","Fry","Thur","Wed","Tue","
Mon"))+

  labs(x="Time of Day (hours)", y="Day of Week")+ coord_fixed()

arr_agg_month    <- aggregate(ArrDelay~DayOfWeek+Month,airports,FUN='sum')

ggplot(arr_agg_month, aes(Month,y=DayOfWeek))+

  geom_tile(aes(fill=ArrDelay))+

  scale_fill_gradientn(colours=brewer.pal(9,"Greys"),

                       breaks=seq(0,max(arr_agg_month$ArrDelay),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()

Dep_agg_month    <- aggregate(DepDelay~DayOfWeek+Month,airports,FUN='sum')

ggplot(Dep_agg_month, aes(Month,y=DayOfWeek))+

  geom_tile(aes(fill=DepDelay))+

  scale_fill_gradientn(colours=brewer.pal(9,"Greys"),

                       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()
```

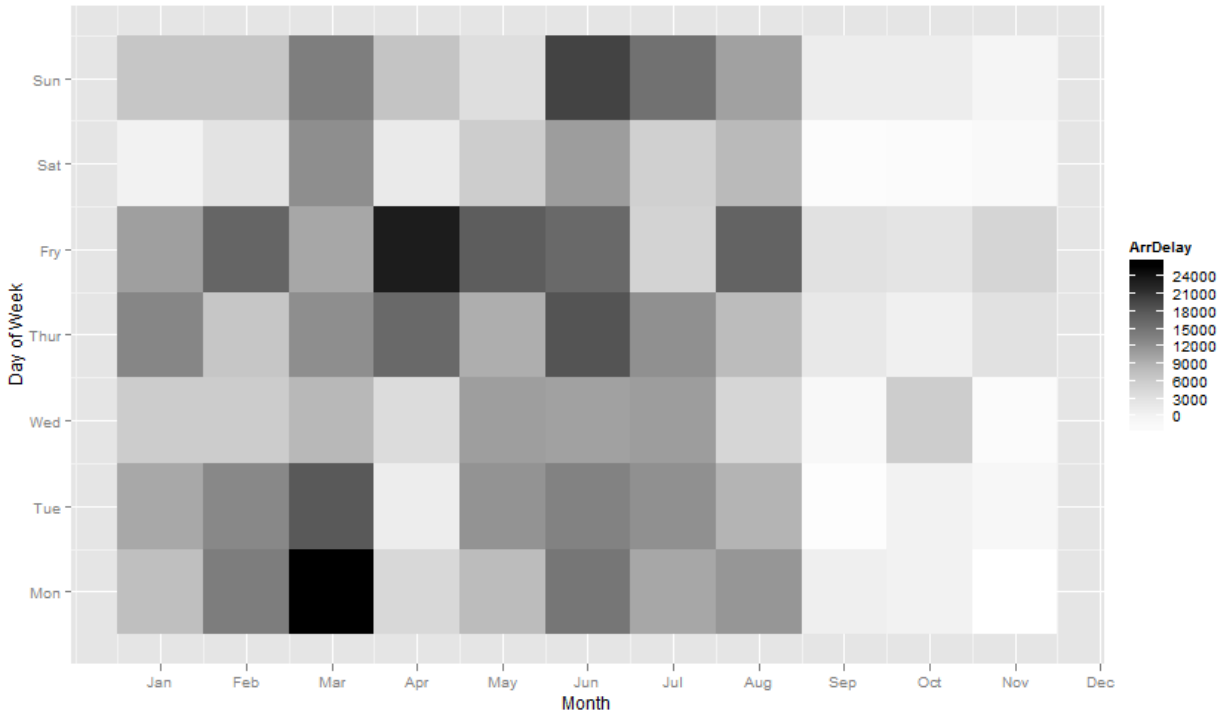
Departure Delays are worst on the Mondays of the month March. The second worst delays are on the Sundays of June.The months of September, October and November are least affected by Departure Delays.



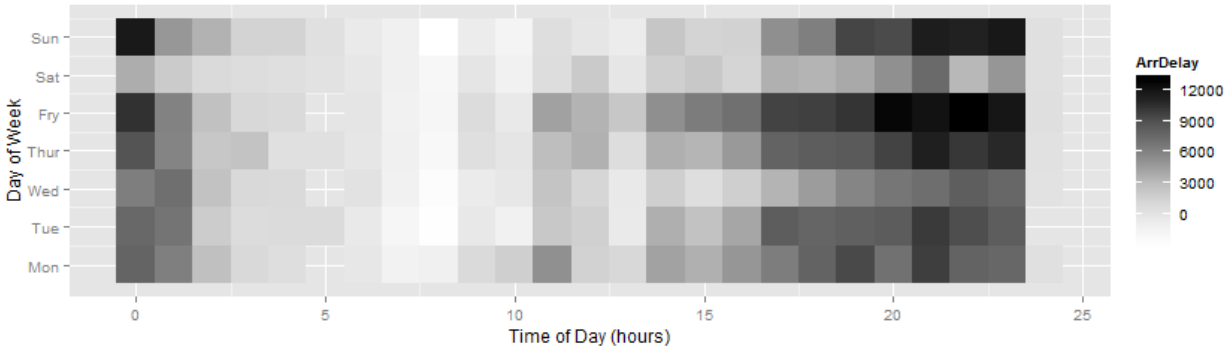
Departure Delays are mostly between 1600 and 2100 hours. The most affected weekdays are Thursdays, Fridays and Sundays



Mondays in the month of March and Fridays in the month of April are the worst effected by the delays in Arrival



Arrival Delays are mostly between 1600 and 2200 hours, a lot similar to the Delay Delays.



Question2: Author Attribution

We Import the libraries. The 'tm' library will used to the text mining commands, eg: tokenization of training and test corpus and readplain command. The 'randomForest' library is used because we run the Random Forest model as one of our methods. The 'e1071' library is used because we are using the Naive Bayes model model.

```
library(tm)

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

## Loading required package: NLP

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

library(randomForest)

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

## 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)

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

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.2.2
```

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

```
library(caret)
```

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

```
## Loading required package: lattice
```

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

We create a Reader function which reads each line through the readPlain function. This function reads in a text document without knowledge about its internal structure and possible available metadata.

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

Now, we create the Training Corpus. The sys.glob function here is a function to do wildcard expansion on file paths.

```
author_dirs =
Sys.glob("https://github.com/jgscott/STA380/tree/master/data/ReutersC50/C50train/*")
file_list = NULL
train_labels = NULL
for(author in author_dirs) {
  author_name = substring(author, first=23)
  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)))
}
```

We perform Named conversion & cleanup of the file. The lapply function here returns a list of the same length as file_list, each element of which is the result of applying readerPlain to the corresponding element of file_list

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

In the next step, we initialize Training Corpus

```
train_corpus = Corpus(VectorSource(all_docs))
names(train_corpus) = file_list
```

Tokenization of training Corpus: Now, in order to tokenize the training corpus, we follow the followin steps: First we convert the content of train_corpus to lowercase. Then, we remove numbers and punctuation. Next, we strip the white spaces

```
train_corpus = tm_map(train_corpus, content_transformer(tolower))
train_corpus = tm_map(train_corpus, content_transformer(removeNumbers))
train_corpus = tm_map(train_corpus, content_transformer(removePunctuation))
train_corpus = tm_map(train_corpus, content_transformer(stripWhitespace))
train_corpus = tm_map(train_corpus, content_transformer(removeWords),
stopwords("SMART"))
```

Create training DTM & dense matrix: Turning this list into a document-term matrix. After that, we remove the sparce terms and include those terms which have a sparce factor of less than 97.5%

```
DTM_train = DocumentTermMatrix(train_corpus)
DTM_train = removeSparseTerms(DTM_train, 0.975)
```

Creating the Testing Corpus. Just like the training corpus, the sys.glob function here is a function to do wildcard expansion on file paths

```
author_dirs =
Sys.glob("https://github.com/jgscott/STA380/tree/master/data/ReutersC50/C50te
st/*")
file_list = NULL
test_labels = NULL
for(author in author_dirs) {
  author_name = substring(author, first=22)
  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)))
}
```

We perform Named conversion & cleanup of the file. The lapply function here returns a list of the same length as file_list, each element of which is the result of applying readerPlain to the corresponding element of file_list

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

Then, we initialize Training Corpus

```
test_corpus = Corpus(VectorSource(all_docs))
names(test_corpus) = file_list
```

Tokenization of training Corpus: Now, in order to tokenize the training corpus, we follow the followin steps: First we convert the content of train_corpus to lowercase. Then, we remove numbers and punctuation. Next, we strip the white spaces

```
test_corpus = tm_map(test_corpus, content_transformer(tolower))
test_corpus = tm_map(test_corpus, content_transformer(removeNumbers))
test_corpus = tm_map(test_corpus, content_transformer(removePunctuation))
test_corpus = tm_map(test_corpus, content_transformer(stripWhitespace))
test_corpus = tm_map(test_corpus, content_transformer(removeWords),
stopwords("SMART"))
```

As we need a dictionary of terms from the training corpus in order to extract terms from the test corpus, we create a Dictionary

```
reuters_dict = NULL
reuters_dict = dimnames(DTM_train)[[2]]
```

Then, we create the testing document term matrix using dictionary words only

```
DTM_test = DocumentTermMatrix(test_corpus, list(dictionary=reuters_dict))
DTM_test = removeSparseTerms(DTM_test, 0.975)
```

In order for it to work in classifier models, we convert DTMs into Data Frames

```
DTM_train_df = as.data.frame(inspect(DTM_train))
```

```
## <<DocumentTermMatrix (documents: 0, terms: 0)>>
## Non-/sparse entries: 0/0
## Sparsity           : 100%
## Maximal term length: 0
## Weighting          : term frequency (tf)
##
## <0 x 0 matrix>
```

```
DTM_test_df = as.data.frame(inspect(DTM_test))
```

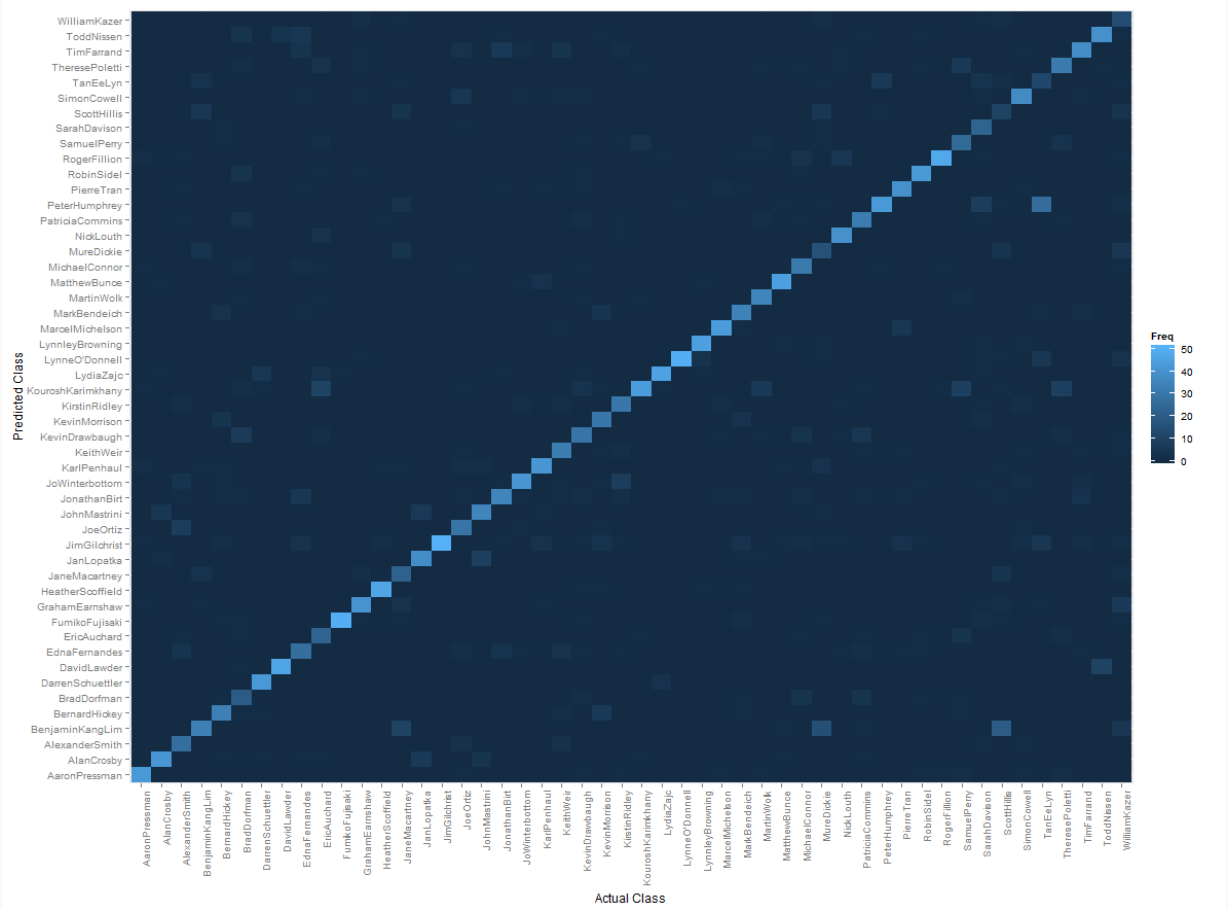
```
## <<DocumentTermMatrix (documents: 0, terms: 0)>>
## Non-/sparse entries: 0/0
## Sparsity           : 100%
## Maximal term length: 0
## Weighting          : term frequency (tf)
##
## <0 x 0 matrix>
```

Random Forest Model

RandomForest requires the same variables in training and test sets. Therefore, we need to add empty columns in the test data set for words that appear in the training data, but not in the test data

```
DTM_test = as.matrix(DTM_test)
```

```
DTM_train = as.matrix(DTM_train)



xx <- data.frame(DTM_test[,intersect(colnames(DTM_test),
colnames(DTM_train))])

yy <- read.table(textConnection(""), col.names = colnames(DTM_train),
colClasses = "integer")

library(plyr)



DTM_test_clean = rbind.fill(xx, yy)

DTM_test_df = as.data.frame(DTM_test_clean)



model_RF = randomForest(x=DTM_train_df, y=as.factor(train_labels), mtry=3,
ntree=200)

pred_RF = predict(model_RF, data=DTM_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))
```

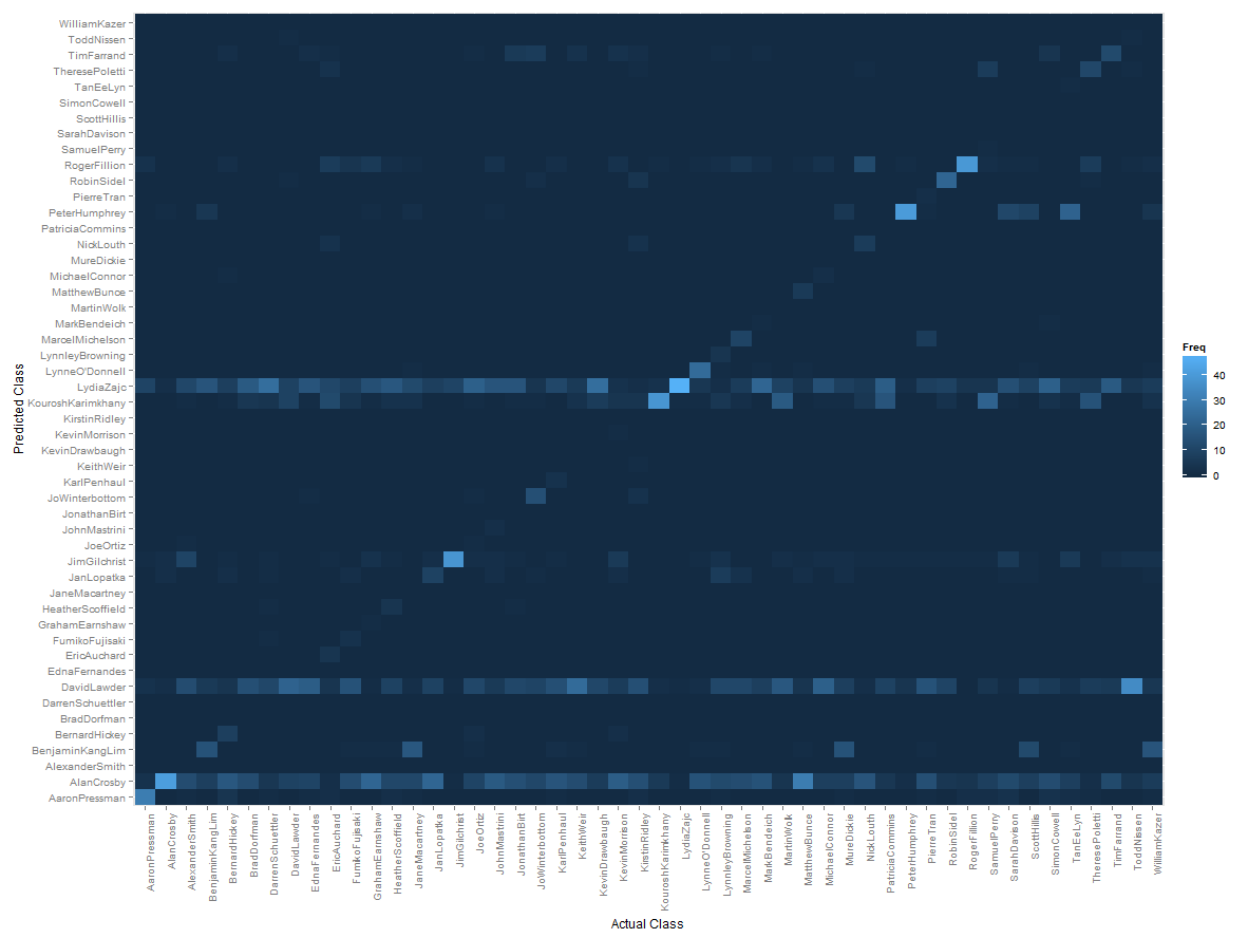



```
conf_RF = confusionMatrix(table(pred_RF,test_labels))
conf_RF$overall
conf_RF_df = as.data.frame(conf_RF$byClass)
conf_RF_df[order(-conf_RF_df$Sensitivity),1:2]
```

Naive Bayes Model

```
model_NB = naiveBayes(x=DTM_train_df, y=as.factor(train_labels), laplace=1)


pred_NB = predict(model_NB, DTM_test_df)
table_NB = as.data.frame(table(pred_NB,test_labels))
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 = confusionMatrix(table(pred_NB,test_labels))
conf_NB_df = as.data.frame(conf_NB$byClass)
conf_NB_df[order(-conf_NB_df$Sensitivity),1:2]
```

After running Naive Bayes and Random Forest models, we conclude that Random Forest, with an accuracy of 69.4% is better than Naive Bayes which has an accuracy of 18%. From the results of the confusion matrix, we can see that the authors whose articles seem difficult from one another are: BradDorfman, EdnaFernandes, JaneMacartney, JonathanBirt, KeithWeir, KevinDrawbaugh, KirstinRidley, KirstinRidley, MureDickie, ScottHillis, WilliamKazer

Question3: Association rule mining

Importing arules library as it has a big ecosystem of packages built around it

```
library(arules)
```

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

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

Read in the groceries file from users

```
groceries1=read.transactions("https://raw.githubusercontent.com/jgscott/STA380/master/data/groceries.txt", format ="basket", sep = ",",
                             rm.duplicates = TRUE)
```

Now, we cast the groceries1 variable as a special arules "transactions" class

```
playtrans <- as(groceries1, "transactions")
```

Now run the 'Apriori' algorithm.

Look at rules with support > .01 & confidence >.5 & length (# artists) <= 4

```
musicrules <- apriori(playtrans,
                      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].
```

Now, we look at the output

```
inspect(musicrules)
```

```
##    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
```

Next, we see which Item-Sets are most likely to occur. We should remember that Higher Lift means higher statistical dependance. We chose 3 because this is the highest value of lift that shows us any subsets

```
inspect(subset(musicrules, 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
```

Let's see that subset where another product occurs at least 58% of the time along with certain products. I chose 58% as this a very strong confidence within this data beaucause a 59% confidence returns no subsets

```
inspect(subset(musicrules, 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
```

Finally, we choose the subset that has the highest values for support and confidence

```
inspect(subset(musicrules, subset=support > .012 & confidence > 0.58))
```

```
##   lhs                    rhs                  support confidence     lift
## 1 {root vegetables,
##    tropical fruit}  => {other vegetables} 0.012303  0.5845411 3.020999
```

Based on the result, we infer that: (a) citrus fruit, root vegetables are bought together with 'other vegetables' (b) root vegetables, tropical fruit is bought together with 'other vegetables' (c) Curd, yogurt are bought along with whole milk (d) citrus fruit, root vegetables are bought along with other vegetables (e) root vegetables, tropical fruit are bought along with other vegetables

These item sets make sense as most of these items are often purchased together in the grocery store