Question 1

Nalini Agrawal

Tuesday, August 18, 2015

Question 1

```
ABIA = read.csv('C:/Users/32inder/Desktop/MSBA/Academic/Predictive
Modelling/STA380/data/ABIA.csv')
```

Data cleaning

```
ABIA[is.na(ABIA)]= 0
attach(ABIA)
```

Convert continuous variables into factors

```
ABIA$DayofMonth= as.factor(ABIA$DayofMonth)
ABIA$DayOfWeek= as.factor(ABIA$DayOfWeek)
ABIA$Month= as.factor(ABIA$Month)
```

Run the aggregate function on the Departure Delay and Arrival Delay columns by Days of the Week. This gives the average arrival and departure delays for each day of the week.

We are also aggregating various metrics like the Security Delay, Carrier Delay, Weather Delay, NAS Delay, LateAircraft Delay for the days of the week and for the months of the year.

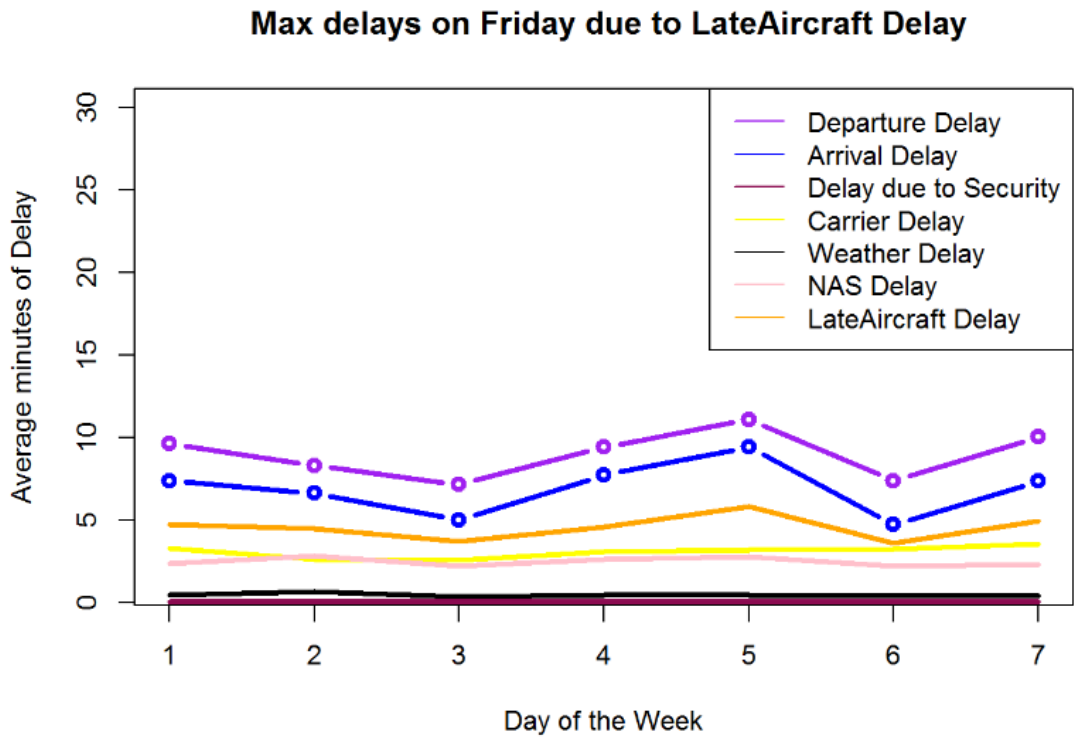
```
aggdepdelay = aggregate(DepDelay,by = list(DayOfWeek), FUN = mean, na.rm=
TRUE)
aggarrdelay = aggregate(ArrDelay,by = list(DayOfWeek), FUN = mean, na.rm=
TRUE)
aggsecdelay = aggregate(SecurityDelay,by = list(DayOfWeek), FUN = mean,
na.rm= TRUE)
aggcardelay = aggregate(CarrierDelay,by = list(DayOfWeek), FUN = mean, na.rm=
TRUE)
aggweatdelay = aggregate(WeatherDelay,by = list(DayOfWeek), FUN = mean,
na.rm= TRUE)
aggNASdelay = aggregate(NASDelay,by = list(DayOfWeek), FUN = mean, na.rm=
TRUE)
agglatefldelay = aggregate(LateAircraftDelay,by = list(DayOfWeek), FUN =
mean, na.rm= TRUE)


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

```
aggarrdelaymonth = aggregate(ArrDelay,by = list(Month), FUN = mean, na.rm=
TRUE)
aggsecdelaymonth = aggregate(SecurityDelay,by = list(Month), FUN = mean,
na.rm= TRUE)
aggcardelaymonth = aggregate(CarrierDelay,by = list(Month), FUN = mean,
na.rm= TRUE)
aggweatdelaymonth = aggregate(WeatherDelay,by = list(Month), FUN = mean,
na.rm= TRUE)
aggNASdelaymonth = aggregate(NASDelay,by = list(Month), FUN = mean, na.rm=
TRUE)
agglatefldelaymonth = aggregate(LateAircraftDelay,by = list(Month), FUN =
mean, na.rm= TRUE)
```

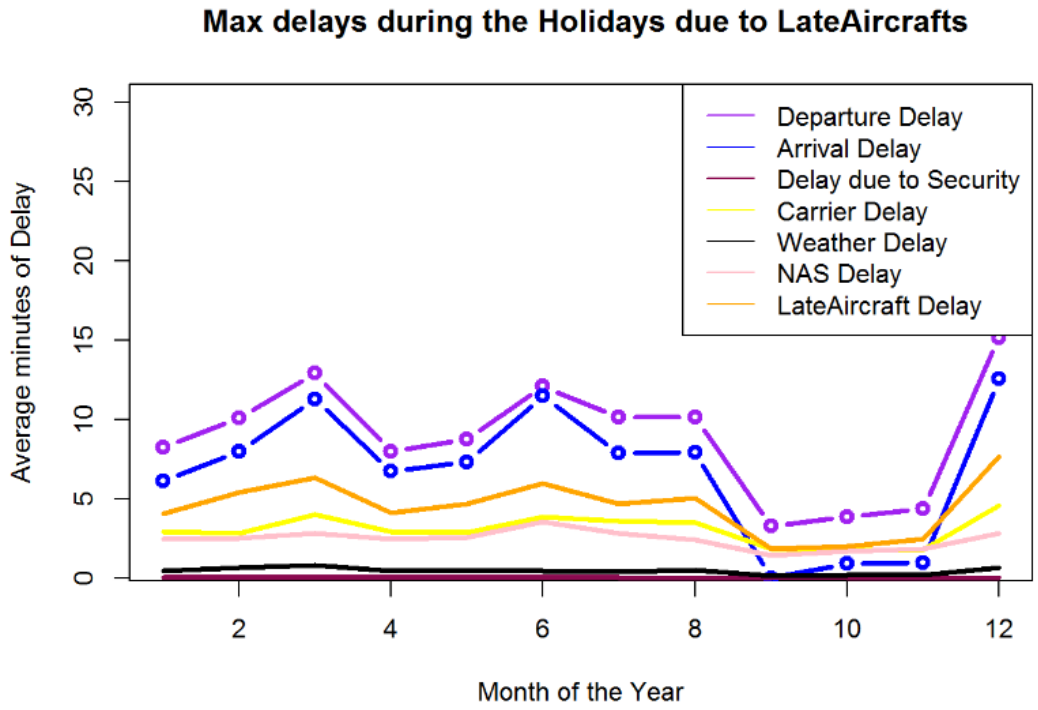
Create a plot for the Average arrival and departure delay for each day of the week and for each Month and adding different measures of delay so analyze which measures lead to the maximum delay.

```
plot(aggdepdelay$x, type ="b", xlab = "Day of the Week", ylab = "Average
minutes of Delay", col = "purple", lwd = 3, ylim = c(1,30), main = " Max
delays on Friday due to LateAircraft Delay" )
lines(aggarrdelay$x, type = "b", col = "blue", lwd = 3)
lines(aggsecdelay$x, type = "l", col = "deeppink4", lwd = 3)
lines(aggcardelay$x, type = "l", col = "yellow", lwd = 3)
lines(aggweatdelay$x, type = "l", col = "black", lwd = 3)
lines(aggNASdelay$x, type = "l", col = "pink", lwd = 3)
lines(agglatefldelay$x, type = "l", col = "orange", lwd = 3)
legend ("topright", c("Departure Delay", "Arrival Delay", "Delay due to
Security", "Carrier Delay", "Weather Delay", "NAS Delay", "LateAircraft
Delay"), lty = 1, col = c('purple','blue', 'deeppink4','yellow', 'black',
'pink', 'orange', xjust = 1, text.width = 2))
```




```
#Creating a plot that shows the aggregate departure, arrival delays, Security
delays, Carrier Delays, Weather Delays, NAS delays, LateAircraft Delays for
each month of the year

plot(aggdepdelaymonth$x, type ="b", xlab = "Month of the Year", ylab =
"Average minutes of Delay", col = "purple", lwd = 3, ylim = c(1,30), main = "
Max delays during the Holidays due to LateAircrafts" )
lines(aggarrdelaymonth$x, type = "b", col = "blue", lwd = 3)
lines(aggsecdelaymonth$x, type = "l", col = "deeppink4", lwd = 3)
lines(aggcardelaymonth$x, type = "l", col = "yellow", lwd = 3)
lines(aggweatdelaymonth$x, type = "l", col = "black", lwd = 3)
lines(aggNASdelaymonth$x, type = "l", col = "pink", lwd = 3)
lines(agglatefldelaymonth$x, type = "l", col = "orange", lwd = 3)
legend ("topright", c("Departure Delay", "Arrival Delay", "Delay due to
Security", "Carrier Delay", "Weather Delay", "NAS Delay", "LateAircraft
Delay"), lty = 1, col = c('purple','blue', 'deeppink4','yellow', 'black',
'pink', 'orange'))
```



We can deduce the following form Plot 1:

- Highest minutes of arrival and departure delays are on Friday
- Measures like the Delay in Security, Weather delay, NAS delay, are consistent which shows that these are not the causes of the delay for these flights
- The LateAircraft Delay closely mimicks the Arrival Delay. This tells us that the key reason for the delay in the flights is LateAircraft Delay.

We can deduce the following form Plot 2:

- The peaks for the arrival and departure delays are in March, June and after November. This shows that the highest delays are during the holiday season.
- Measures like the Delay in Security, Weather delay, NAS delay, are consistent which shows that these are not the causes of the delay for these flights
- The LateAircraft Delay closely mimicks the Arrival Delay. This tells us that the key reason for the delay in the flights is LateAircraft Delay.

Question 2

Question 2

```
#Importing the library to run the different models for this question
library(tm)

## Warning: package 'tm' was built under R version 3.2.1

## Loading required package: NLP

## Warning: package 'NLP' was built under R version 3.2.1

library(randomForest)

## Warning: package 'randomForest' was built under R version 3.2.1

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

library(rpart)
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.2.1

##
## 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

library(plyr)

## Warning: package 'plyr' was built under R version 3.2.1

library(e1071)

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

Sourcing the Reader Plain function that will act as a helper function when importing the data

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

Create the Training Corpus

```
author_dirs = Sys.glob('C:/Users/32inder/Desktop/MSBA/Academic/Predictive
Modelling/STA380/data/ReutersC50/C50train/*')
file_list = NULL
train_labels = NULL
for(author in author_dirs) {
  author_name = substring(author, first=93)
  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)))
}
```

Clean up the data and adding better names to the training corpus

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

Initialize the Training corpus

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

Pre-process the data and tokenizing it

```
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"))
train_corpus = tm_map(train_corpus, content_transformer(tolower))
```

Create the training Document Term Matrix and a dense matrix

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

Create the Testing Corpus

```
author_dirs = Sys.glob('C:/Users/32inder/Desktop/MSBA/Academic/Predictive
Modelling/STA380/data/ReutersC50/C50test/*')
file_list = NULL
test_labels = NULL
for(author in author_dirs) {
  author_name = substring(author, first=92)
  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)))
}
```

Clean up the data and add better names to the testing corpus

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

Initialize the testing corpus

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

Pre-processing the data and tokenizing it

```
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"))
```

Create a Dictionary

We need to create a dictionary since there are words in the test set that we have not seen in the training set. This dictionary will allow us to take words from the testing corpus.

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

Using the words in the dictionary create a Document Term Matrix and matrix. This allows us to mitigate the effect of words that are in the testing corpus but not in the training corpus.

```
DTM_test = DocumentTermMatrix(test_corpus, list(dictionary=rdict))
DTM_test = removeSparseTerms(DTM_test, 0.95)
```

Convert a Document Term Matrix into a DataFrame so it can be used in a Classifier Model

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

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

Using the Naive Bayes Model to predict the authors

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

Use the Naive Bayes model to predict on the test corpus

```
naivebayespredict = predict(naivebayes, DTM_test_df)
```

Tabulate the results and generate summary statistics.

```
confusionmatNB = confusionMatrix(table(naivebayespredict, test_labels))
confusionmatNB$overall

##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.2580000      0.2428571      0.2409405      0.2756282      0.0200000
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

The results from the confusion matrix show that the accuracy score of the model which is 25.8%. Witht this model, we can predict the authors Thus we can see that Naive Bayes is not a very good model for this corpus.

Run a Random Forest model to predict the authors

Convert the test and training Document Term matrices to a matrices

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

Since this model requires the same number of columns in the test and training corpus, we add additional empty columns to the test dataset to ensure that there is an alignment.

```
count_of_words <- data.frame(DTM_test[,intersect(colnames(DTM_test),
colnames(DTM_train))])
words <- read.table(textConnection(""), col.names = colnames(DTM_train),
colClasses = "integer")
```

Bind the x and y data frame and table created above and convert it to a data frame

```
DTM_test_new = rbind.fill(count_of_words, words)

DTM_test_dataframe = as.data.frame(DTM_test_new)
```

Run the Random Forest model to determine the authors

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

Use the above model to Predict

```
randomforestpredict = predict(randomforestmodel, data=DTM_test_clean)
```

Create a confusion matrix to tabulate the results and see the accuracy

```
confusionmatRF = confusionMatrix(table(randomforestpredict, test_labels))
confusionmatRF$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.6976000      0.6914286      0.6791720      0.7155641      0.0200000
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

The above confusion matrix gives us an accuracy score of 69.76% after using a Random Forest model. This shows that Random Forest is a better model to use for this particular dataset.

The Naive Bayes model does not do well in this model since I chose a sparsity of 95%. If we decrease the sparsity of the model, the accuracy will improve but the model will be overfitting. With the metrics that we chose above, Random Forest provides a good accuracy score without overfitting the model.

Question 3

Question 3

```
library(arules)  # has a big ecosystem of packages built around it
```

```
## 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
groceries <-
read.transactions('https://raw.githubusercontent.com/jgscott/STA380/master/da
ta/groceries.txt', format = 'basket', sep = ',')
```

Applying the apriori algorithm to find itemsets that occur frequently

```
groceriesrules <- 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 [0.01s].
## checking subsets of size 1 2 3 4 done [0.01s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
# Look at the output by using the inspect function
inspect(groceriesrules)
```

```
##    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 to inspect the data

```
#Different subsets using different metrics give different results:


#When lift >3:
inspect(subset(groceriesrules, 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
```

```
# I used lift>3, since this was the highest value of lift that generated the
most exclusive subsets. Higher value of lift would lead to higher dependance,
so taking a high value of lift would show the subsets that have the highest
occurance in the itemsets.

#When confidence > 0.575:

inspect(subset(groceriesrules, subset=confidence > 0.575))
```

```
##   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
```

```
# I used confidence greater than 0.575 since this was the highest value of
confidence, that generated the most exclusive and good subset. A  confidence
higher than 0.59 would not generate any subsets and a confidence much lower
than 0.575 would generate too many subsets. A Higher confidence will show how
often items in whole milk and other vegetables appear in curd, yogurt, root
vegetables, tropical fruit, citrus fruit.
```

```
#When confidence >0.57 and support >0.011:

inspect(subset(groceriesrules, subset=support > .011 & confidence > 0.58))

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

#I used the above two specific metrics as these two metrics combined give us
the most exclusive subset. Support less than 0.011 and confidence lesser than
0.57 would not give us the most exclusive subset with these two metrics.
```