Q1, Assignment 2, ABIA EDA Visualization

Zahref Beyabani

August 16, 2015

An excellent way to better understand the underlying patterns of one of my favorite travel hubs is to take a data-science based approach and perform some EDA using cool graphics.

The airport being examined is the Austin Bergstorm International Airport, in Austin, TX

First, lets read & load the data in 2008 about all flight particulars for this airport

```
#https://raw.githubusercontent.com/jgscott/STA380/master/data/
flight = read.csv("~/Downloads/ABIA.csv")
```

To begin our exploratory analysis, lets see the number of flights delayed in 2008 in ABIA

Before that, some data cleaning & treatment:

```
# Fill NA values in the Delay Columns with 0's to better handle the dat
a
flight[is.na(flight)] <- 0

# Create those cheeky litte factor variables
flight$DayofMonth = as.factor(flight$DayofMonth)
flight$DayOfWeek = as.factor(flight$DayOfWeek)
flight$Month = as.factor(flight$Month)

attach(flight)
```

Group By the Day of the Week and the average departure / arrival delay

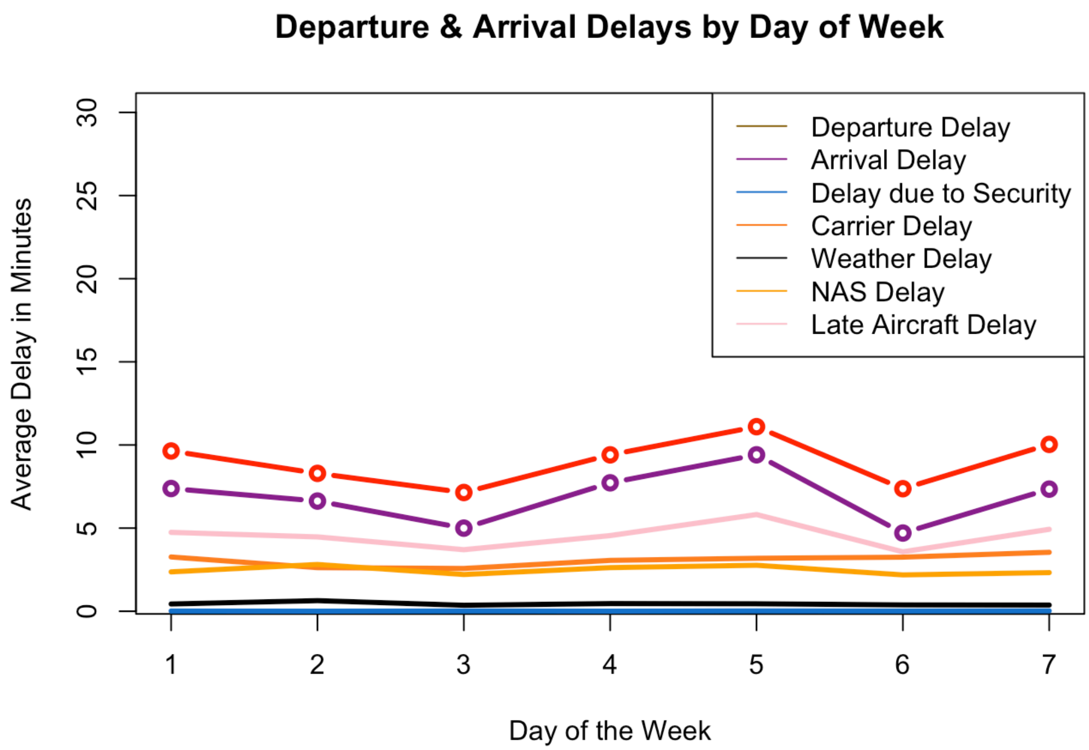
```
avgdepdelay = aggregate(DepDelay,by = list(DayOfWeek), FUN = mean, na.r
m= TRUE)
avgarrdelay = aggregate(ArrDelay,by = list(DayOfWeek), FUN = mean, na.r
m= TRUE)
avgsecdelay = aggregate(SecurityDelay,by = list(DayOfWeek), FUN = mean,
na.rm= TRUE)
avgcardelay = aggregate(CarrierDelay,by = list(DayOfWeek), FUN = mean,
na.rm= TRUE)
avgweatdelay = aggregate(WeatherDelay,by = list(DayOfWeek), FUN = mean,
na.rm= TRUE)
avgNASdelay = aggregate(NASDelay,by = list(DayOfWeek), FUN = mean, na.r
m= TRUE)
avglatefldelay = aggregate(LateAircraftDelay,by = list(DayOfWeek), FUN
```

```
= mean, na.rm= TRUE)


#Aggregating the departure and arrival delays by month of the year
avgdepdelaymonth = aggregate(DepDelay,by = list(Month), FUN = mean, na.
rm= TRUE)
avgarrdelaymonth = aggregate(ArrDelay,by = list(Month), FUN = mean, na.
rm= TRUE)
avgsecdelaymonth = aggregate(SecurityDelay,by = list(Month), FUN = mean
, na.rm= TRUE)
avgcardelaymonth = aggregate(CarrierDelay,by = list(Month), FUN = mean,
na.rm= TRUE)
avgweatdelaymonth = aggregate(WeatherDelay,by = list(Month), FUN = mean
, na.rm= TRUE)
avgNASdelaymonth = aggregate(NASDelay,by = list(Month), FUN = mean, na.
rm= TRUE)
avglatefldelaymonth = aggregate(LateAircraftDelay,by = list(Month), FUN
= mean, na.rm= TRUE)
```

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

```
plot(avgdepdelay$x, type ="b", xlab = "Day of the Week", ylab = "Averag
e Delay in Minutes", col = "red", lwd = 3, ylim = c(1,30), main = "Depa
rture & Arrival Delays by Day of Week" )
lines(avgarrdelay$x, type = "b", col = "darkmagenta", lwd = 3)
lines(avgsecdelay$x, type = "l", col = "dodgerblue3", lwd = 3)
lines(avgcardelay$x, type = "l", col = "chocolate1", lwd = 3)
lines(avgweatdelay$x, type = "l", col = "black", lwd = 3)
lines(avgNASdelay$x, type = "l", col = "orange", lwd = 3)
lines(avglatefldelay$x, type = "l", col = "pink", lwd = 3)
legend ("topright", c("Departure Delay", "Arrival Delay", "Delay due to
Security", "Carrier Delay", "Weather Delay", "NAS Delay", "Late Aircraf
t Delay"), lty = 1, col = c('darkgoldenrod4','darkmagenta', 'dodgerblue
3','chocolate1', 'black', 'orange', 'pink'))
```

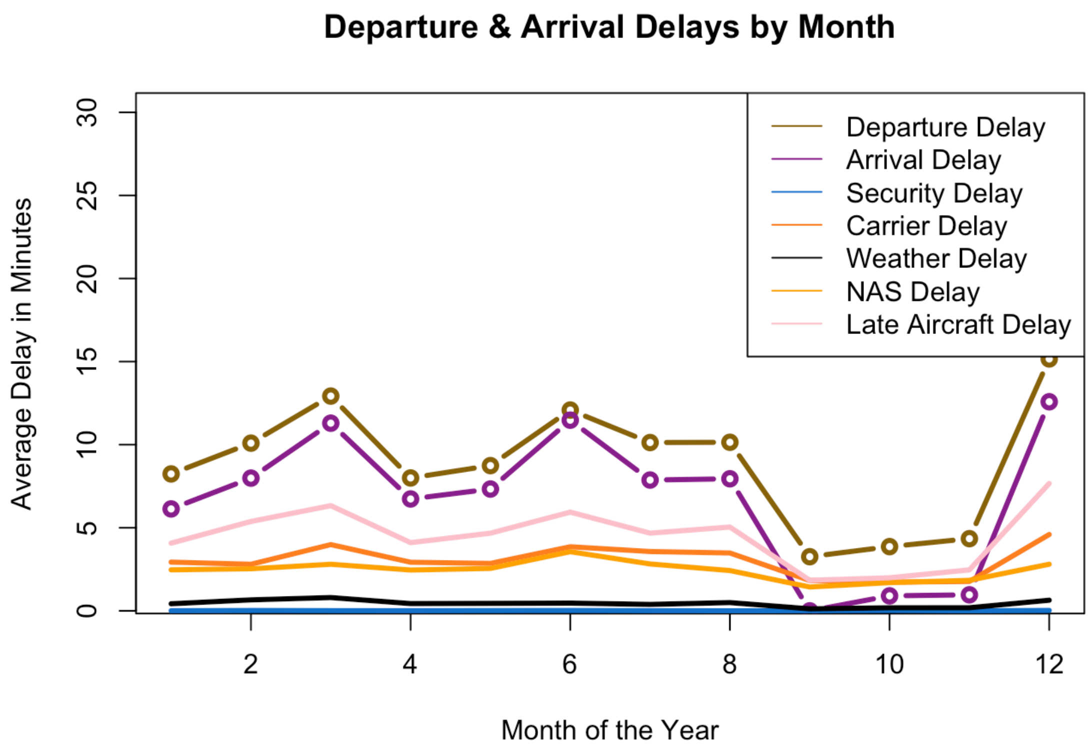



```r
# now for month

plot(avgdepdelaymonth$x, type ="b", xlab = "Month of the Year", ylab =
"Average Delay in Minutes", col = "darkgoldenrod4", lwd = 3, ylim = c(1
,30), main = "Departure & Arrival Delays by Month")

lines(avgarrdelaymonth$x, type = "b", col = "darkmagenta", lwd = 3)
lines(avgsecdelaymonth$x, type = "l", col = "dodgerblue3", lwd = 3)
lines(avgcardelaymonth$x, type = "l", col = "chocolate1", lwd = 3)
lines(avgweatdelaymonth$x, type = "l", col = "black", lwd = 3)
lines(avgNASdelaymonth$x, type = "l", col = "orange", lwd = 3)
lines(avglatefldelaymonth$x, type = "l", col = "pink", lwd = 3)

legend ("topright", c("Departure Delay", "Arrival Delay", "Security Del
ay", "Carrier Delay", "Weather Delay", "NAS Delay", "Late Aircraft Dela
y"), lty = 1, col = c('darkgoldenrod4','darkmagenta', 'dodgerblue3','ch
ocolate1', 'black', 'orange', 'pink'))
```



* We see that we have the highest average delay on Friday.

- We see that we have the highest average delay by month in Decemeber, June, and March.
  - This is expected as these are popular holiday seasons and we expect higher passenger volume during these periods.
- Across both aggreagations, month-wise & day-wise, Security, Weather, and NAS do not contribute to most of the delay. However, most of the delay comes from the late arrival of the incoming aircraft, and carrier based delays.

Q2 Assginment2

Zahref Beyabani

August 17, 2015

```
library(tm)

## Loading required package: NLP

library(randomForest)
```

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

library(e1071)
library(rpart)
library(ggplot2)

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

library(caret)

## Loading required package: lattice

library(plyr)
```

Sourcing the reader plain function

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

Creating the training corpus

```
author_dirs = Sys.glob('STA380/data/ReutersC50/C50train/*')
author_dirs = author_dirs[1:50]
file_list = NULL
tr_labels = NULL
for(author in author_dirs) {
  author_name = substring(author, first=33)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  tr_labels = append(tr_labels, rep(author_name, length(files_to_add)))
}
all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))

my_corpus = Corpus(VectorSource(all_docs))
names(my_corpus) = file_list

# Preprocessing
my_corpus = tm_map(my_corpus, content_transformer(tolower)) # make ever
ything lowercase
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers)) # rem
ove numbers
my_corpus = tm_map(my_corpus, content_transformer(removePunctuation)) #
remove punctuation
```

```
my_corpus = tm_map(my_corpus, content_transformer(stripWhitespace)) ##
remove excess white-space
my_corpus = tm_map(my_corpus, content_transformer(removeWords), stopwor
ds("SMART"))
```

Creating a document-term-matrix & Dense Matrix for the training corpus

```
DTM_tr = DocumentTermMatrix(my_corpus)
DTM_tr = removeSparseTerms(DTM_tr, 0.935)
```

Testing data

Creating the test corpus

```
author_dirs = Sys.glob('STA380/data/ReutersC50/C50test/*')
file_list = NULL
test_labels = NULL
for(author in author_dirs) {
  author_name = substring(author, first=32)
  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_ad
d)))
}
all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))

my_corpus_test = Corpus(VectorSource(all_docs))
names(my_corpus_test) = file_list

# Preprocessing
my_corpus_test = tm_map(my_corpus_test, content_transformer(tolower)) #
make everything lowercase
my_corpus_test = tm_map(my_corpus_test, content_transformer(removeNumbe
rs)) # remove numbers
my_corpus_test = tm_map(my_corpus_test, content_transformer(removePunct
uation)) # remove punctuation
my_corpus_test = tm_map(my_corpus_test, content_transformer(stripWhites
pace)) ## remove excess white-space
my_corpus_test = tm_map(my_corpus_test, content_transformer(removeWords
), stopwords("SMART"))
```

Creating a document-term-matrix & Dense Matrix for the test corpus

```
DTM_test = DocumentTermMatrix(my_corpus_test)
DTM_test = removeSparseTerms(DTM_test, 0.935)
```

Dictionary Creation

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

#Create testing DTM & matrix using dictionary words only
DTM_test = DocumentTermMatrix(my_corpus_test, list(dictionary=reuters_d
ictionary))
DTM_test = removeSparseTerms(DTM_test, 0.935)
#DTM_test = as.matrix(DTM_test)
```

Convert DTM into Data Frames for use in classification models

```
DTM_tr_df = as.data.frame(inspect(DTM_tr))
#DTM_train$auth_name = train_labels
DTM_test_df = as.data.frame(inspect(DTM_test))
#DTM_test$auth_name = test_labels
```

Lets Run a Naïve Bayes Model

```
nb_mod = naiveBayes(x=DTM_tr_df, y=as.factor(tr_labels), laplace=1)
```

Lets run predictions on this model

```
preditions_nb_mod = predict(nb_mod, DTM_test_df)
```

Lets cast these results into a table along with the *ACTUAL* author

```
table_nb_mod = as.data.frame(table(preditions_nb_mod,test_labels))
```

See how often we are wrong

We will first create a confusion matrix

```
nb_confusion = confusionMatrix(table(preditions_nb_mod,test_labels))
```

Next we will view the statistics from this confusion matrix

```
nb_confusion$overall

##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   Accura
cyNull
##      0.2932000      0.2787755      0.2754061      0.3114796      0.0
200000
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

- The accuracy is 29% which means on average we will predict the author with a 29% accuracy when given a sample of writing from the R50 corpus

Lets Run a Random Forest Model

Cast the DTMs to regular matrices so the rf package can interpret it

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

Oops! Random Forests *NEEDS* the number of columns in the training and test matricies to be the same. Let's add additional empty columns (for words we havent seen) into the test dataset to coerce alignment.

```
word_counts = data.frame(DTM_test[,intersect(colnames(DTM_test), colnam
es(DTM_tr))])
col_names = read.table(textConnection(""), col.names = colnames(DTM_tr)
, colClasses = "integer")
```

Bind the word counts along with their names

```
DTM_test_scrubbed = rbind.fill(word_counts, col_names)
DTM_test_df = as.data.frame(DTM_test_scrubbed)
```

Model this data using a random forest

```
rf_mod = randomForest(x=DTM_tr_df, y=as.factor(tr_labels), mtry=4, ntre
e=200)
```

Predict using this model

```
rf_mod_predictions = predict(rf_mod, data=DTM_test_scrubbed)
```

Lets see how this model did!

```
rf_confusion = confusionMatrix(table(rf_mod_predictions,test_labels))
rf_confusion$overall
```

- This random forest model gives us a 69% accuracy which means on average we will predict the author with a 69% accuracy when given a sample of writing from the R50 corpus

IN CONCLUSION:

The Naïve Bayes prediction model does not do as good of a job as the Random Forest classifier when presented with articles from the R50 corpus to attribute to a known author.

Q3, Assignment2

Zahref Beyabani

August 17, 2015

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

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

# Read
groceries <- read.transactions("STA380/data/groceries.txt", format = 'b
asket', sep = ',')
```

Applying the "apriori" algorithm to find frequent item-sets

```
groceriesrules <- apriori(groceries, parameter=list(support=.01, confid
ence=.5, maxlen=5))

##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen ma
xlen
##         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 [0.01s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

# Look at the output
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
```

Choosing a subset to inspect the data:

Choose a subset

Different subsets on different parameters show different item-sets

Lets see which Item-Sets are most likely to occur. Recall that Higher Lift means higher statistical dependance

```
inspect(subset(groceriesrules, subset=lift > 3))

##   lhs                     rhs                      support  confidence
lift
## 1 {citrus fruit,
##    root vegetables} => {other vegetables} 0.01037112  0.5862069 3.02
9608
## 2 {root vegetables,
##    tropical fruit}  => {other vegetables} 0.01230300  0.5845411 3.02
0999
```

- I chose 3 up there because this is the highest value of lift that shows us any subsets

Lets see the 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(groceriesrules, subset=confidence > 0.58))

##   lhs                     rhs                      support  confidence
lift
## 1 {curd,
##    yogurt}          => {whole milk}       0.01006609  0.5823529 2.27
9125
## 2 {citrus fruit,
##    root vegetables} => {other vegetables} 0.01037112  0.5862069 3.02
9608
## 3 {root vegetables,
##    tropical fruit}  => {other vegetables} 0.01230300  0.5845411 3.02
0999
```

Lets choose the subset that has the highest exclusivity by choosing the highest values for support and confidence

```
inspect(subset(groceriesrules, subset=support > .012 & confidence > 0.5
8))

##   lhs                     rhs                    support confidence     li
ft
## 1 {root vegetables,
```

```
##    tropical fruit}  => {other vegetables} 0.012303  0.5845411 3.0209
99
```