# Exercise 2

Bonfante

August 14, 2015

## Flights at ABIA

Here we will create a plot to display which months have the highest rate of cancellations. Therefore it will not simply be a count of the cancellations per month but rather a ratio of the total cancellation divided by the total flights in that month in order to give more descriptive results.

```
library(ggplot2)
#Read in the data
flights = read.csv('ABIA.csv', header = TRUE)
#select 'month' and 'Cancelled' columns
flights = flights[,c(2,22)]

#sum cancellations grouped by month
aggflights = aggregate(Cancelled ~ Month ,flights, sum)
class(aggflights)
```

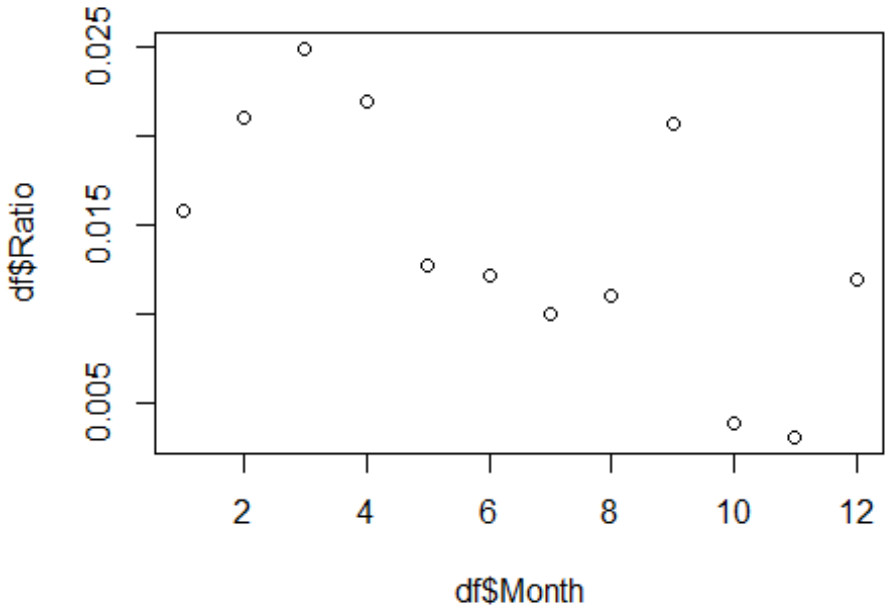
```
## [1] "data.frame"
```

```
#create a count function
count = function(x) {
  length(x)
}

#count the total flights per month and add it to a new column
flightspermonth = aggregate(Cancelled ~ Month ,flights, count)
flightspermonth$Totalflights = flightspermonth$Cancelled
df = cbind.data.frame(aggflights, flightspermonth$Totalflights)

#Rename the column so it looks cleaner
names(df)[3] = paste('Totalflights')
df
```

```
##    Month Cancelled Totalflights
## 1      1       138         8726
## 2      2       171         8156
## 3      3       222         8921
## 4      4       185         8458
## 5      5       115         9021
## 6      6       111         9090
## 7      7        90         8931
```

```
## 8       8         95          8553
## 9       9        154          7464
## 10     10         30          7672
## 11     11         22          7020
## 12     12         87          7248
```

```
#Divide Total cancellations per month by total flights and make it a new
column called 'Ratio'
df = transform(df, Ratio = Cancelled / Totalflights)

#Plot 'Ratio' on y-axis and 'Month' on x-axis
plot(df$Month, df$Ratio)
```




Now we can see which months have the highest Cancellation percentage. The High is in March with over 2.5% of flights being cancelled, and the minimum is during November with less than .5% of flights being cancelled. As the year progresses cancellations tend to decrease other than a large spike in cancellations during the month of September.

## Author Attribution

First import libraries and define our reader function

```
#Import Libraries
library(tm)

## Loading required package: NLP
##
```

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

```
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)
library(caret)
```

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

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

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

First we will create our training matrix

```
author_dirs = Sys.glob('./ReutersC50/C50train/*')
file_list = NULL #list of file directories
train_labels = NULL #List of author names
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)))
}

# clean names
all_docs = lapply(file_list, readerPlain) #Read in all docs
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

#Tokenization of training Corpus
#all lower
train_corpus = tm_map(train_corpus, content_transformer(tolower))
#No numbers
```

```
train_corpus = tm_map(train_corpus, content_transformer(removeNumbers))
#No puncuation
train_corpus = tm_map(train_corpus, content_transformer(removePunctuation))
#No whitespace
train_corpus = tm_map(train_corpus, content_transformer(stripWhitespace))
#No Stopwords
train_corpus = tm_map(train_corpus, content_transformer(removeWords),
stopwords("SMART"))

#Create training DTM
DTM_train = DocumentTermMatrix(train_corpus)
class(DTM_train)
```

```
## [1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

```
DTM_train = removeSparseTerms(DTM_train, 0.96)
DTM_train_matrix = as.matrix(DTM_train)
```

Here we will create the testing corpus

```
author_dirs = Sys.glob('./ReutersC50/C50test/*')
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)))
}

# Clean names
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
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 to pull column names from training matrix
train_names_dict = NULL
```

```
train_names_dict = dimnames(DTM_train)[[2]]
class(train_names_dict)
```

```
## [1] "character"
```

```
#Create testing DTM & matrix with training words only
DTM_test = DocumentTermMatrix(test_corpus, list(dictionary=train_names_dict))
#DTM_test = removeSparseTerms(DTM_test, 0.975)
DTM_test_matrix = as.matrix(DTM_test)

#confirm that DTM_test & DTM_train have the same column names
x = colnames(DTM_train)
y = colnames(DTM_test)
c = cbind(x,y)
#c
```

Now that we have defined and cleaned both our training and testing matrices we can run Naive Bayes and calculate the models accuracy at predicting document authors.

```
#Create the model
NB_model = naiveBayes(x = DTM_train_matrix, y = as.factor(train_labels))
#Use the model to get prediction values
pred = predict(NB_model, DTM_test_matrix)
#Use confus]ion matrix to measure model accuracy
confusion_matrix = confusionMatrix(table(pred, train_labels))
confusion_matrix$overall
```

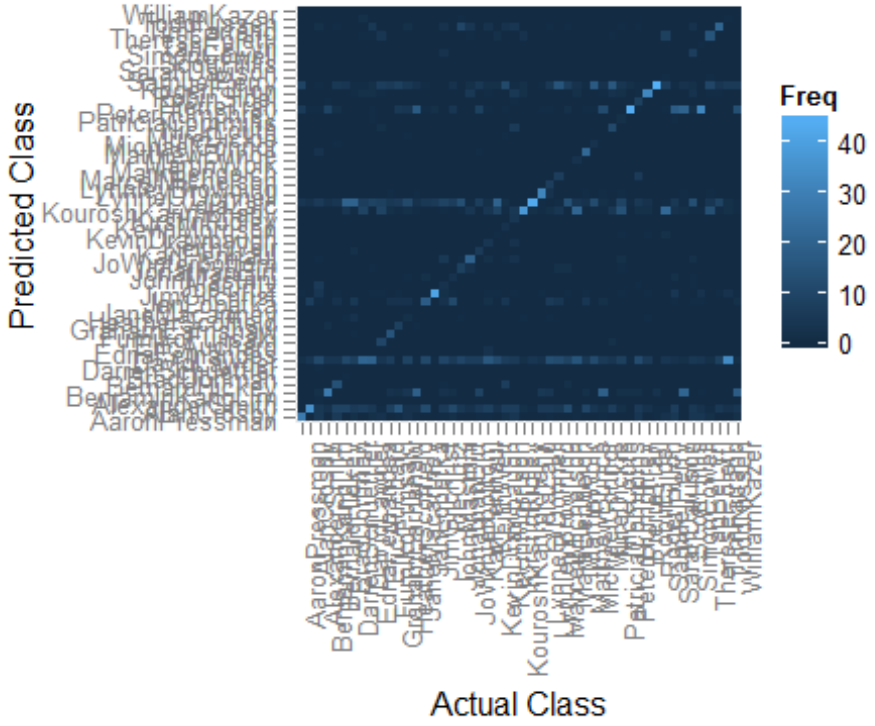
```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.2460000      0.2306122      0.2292231      0.2633739      0.0200000
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
table_NB = as.data.frame(table(pred,train_labels))
```

After running Naive Bayes with Sparsity set to 96%, we got a model accuracy of 24.6%. Although this is not incredible accuracy it is not terrible, and could possibly be improved by using different values for word sparsity.

Plot the Naive Bayes Model

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



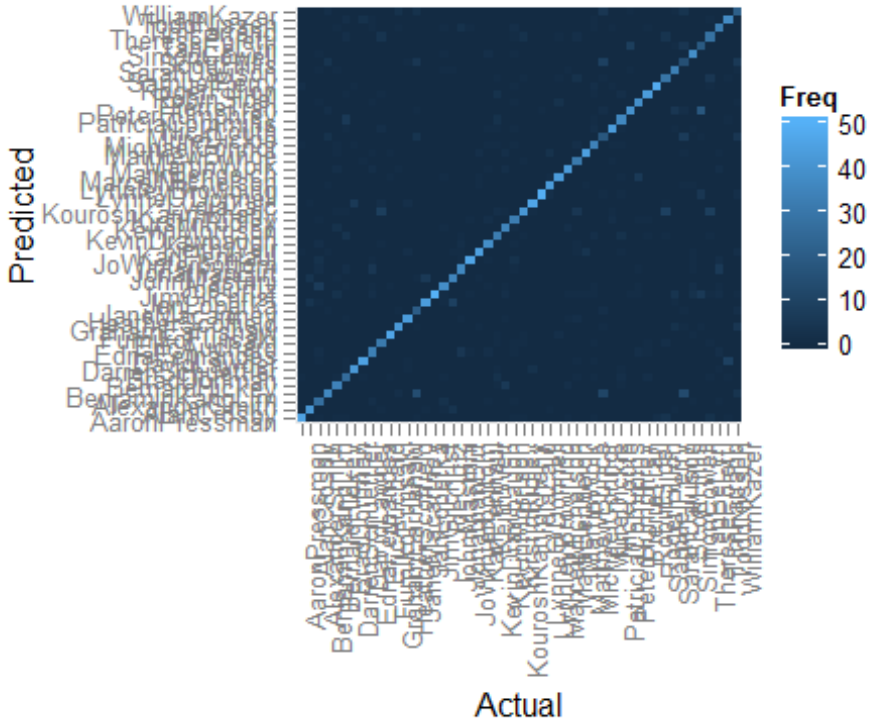

By looking at the calculated plot, you can see that it is dificult to distinguish Roger Filion, Lynn O'donnel, and Edna Fernandez's documents from other others as they were consistently incorrectly predicted.

Now we try to predict authors using a random Forest Model.

```
randomforest = randomForest(x= DTM_train_matrix, y= as.factor(train_labels),
mtry = 5, ntree=200)

rfpredict = predict(randomforest, data = DTM_test_matrix)

confusionrf = confusionMatrix(table(rfpredict, test_labels))
confusionrf$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.7320000      0.7265306      0.7141728      0.7492821      0.0200000
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
table_RF = as.data.frame(table(rfpredict,train_labels))
rfplot = ggplot(table_RF)
rfplot + geom_tile(aes(x=train_labels, y=rfpredict, fill=Freq)) +
    scale_x_discrete(name="Actual") +
    scale_y_discrete(name="Predicted") +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




The Random forrest model performed MUCH beter than Naive Bayes with 69.7% accuracy. Since our Matrices already had the same word columns we were able to quicky write the code to compute RF. We allowed 5 words per decision tree, 200 times. This time the graph shows a much cleaned prediction through the diagonal line. The scattered light squares around this line are icorrect predictions.

## Practice with association rule mining

```
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 data
groceries <- read.transactions("groceries.txt", format = 'basket', sep = ',')
```

```r
#Run apriori on dataset
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 [0.01s].
## 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].
```

```r
# Look at the output
arules::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
```

```
#inspect using different arguments
#arules::inspect(subset(groceriesrules, subset=lift > 2))
#arules::inspect(subset(groceriesrules, subset=confidence > 0.5))
arules::inspect(subset(groceriesrules, subset=support > .01 & confidence >
0.5))
```

```
##    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}             => {whole milk}       0.01453991  0.5629921
2.203354
## 12 {rolls/buns,
##     root vegetables}    => {other vegetables} 0.01220132  0.5020921
2.594890
## 13 {rolls/buns,
##     root vegetables}    => {whole milk}       0.01270971  0.5230126
2.046888
## 14 {other vegetables,
##     yogurt}             => {whole milk}       0.02226741  0.5128806
2.007235
```

After playing around with different pareameters we were able to find a balance in which a concise set of rules were shown. These for the most part seem to make sense especially the first five which say there is high confidence when group various dairy product with whole milk as one might naturally expect.