STA380 – Homework2 – Suraj Tata Prasad

# Question 1 - Flights at ABIA

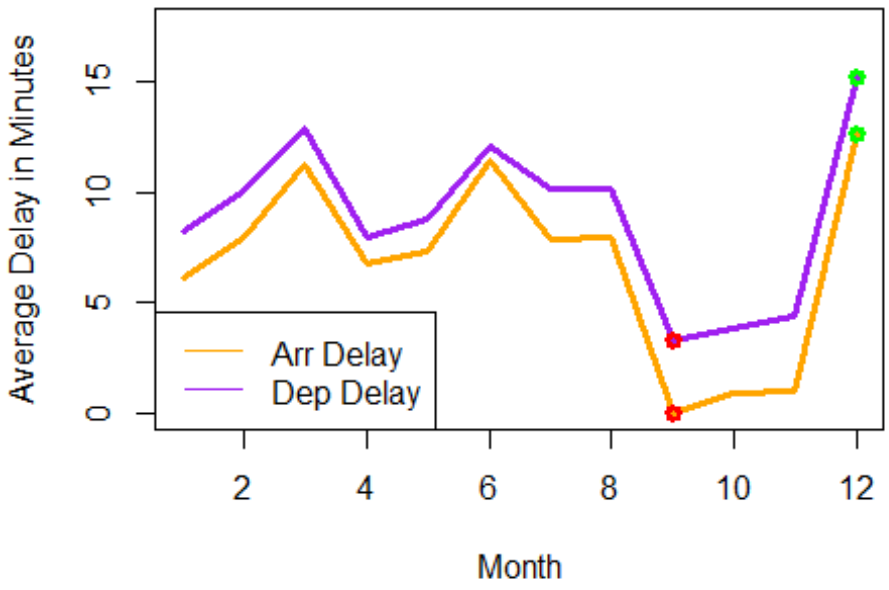
## Imported the dataset and loaded the dplyr library that helps with data manipulations

## Cheking for the worst time in the year to fly by plotting the average delay time in minutes across all months

- Converting the 'days of the month' and 'months of the year' into factors. They are currently continuous variables.
- Also replacing 'NA's in the dataset with '0' to avoid data wrangling issues

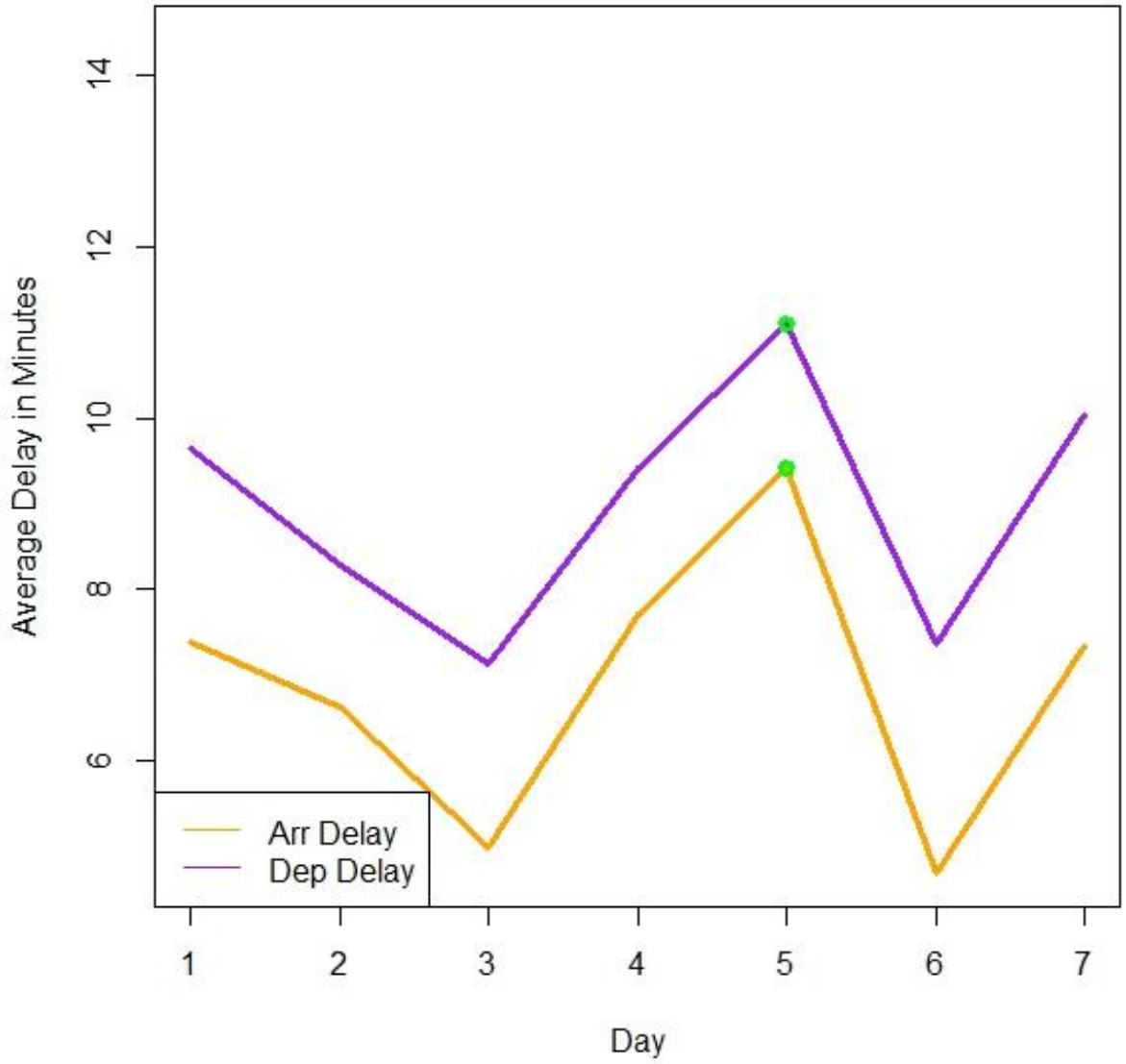
```
flights2$DayofMonth = as.factor(flights2$DayofMonth)
flights2$DayOfWeek = as.factor(flights2$DayOfWeek)
flights2$Month = as.factor(flights2$Month)
flights2[is.na(flights2)]= 0
```

- Considering only arrival and departure delays initially to evaluate the worst time to fly
- Creating a trendline showing the average delay across the months and highlighting the worst and best times to fly

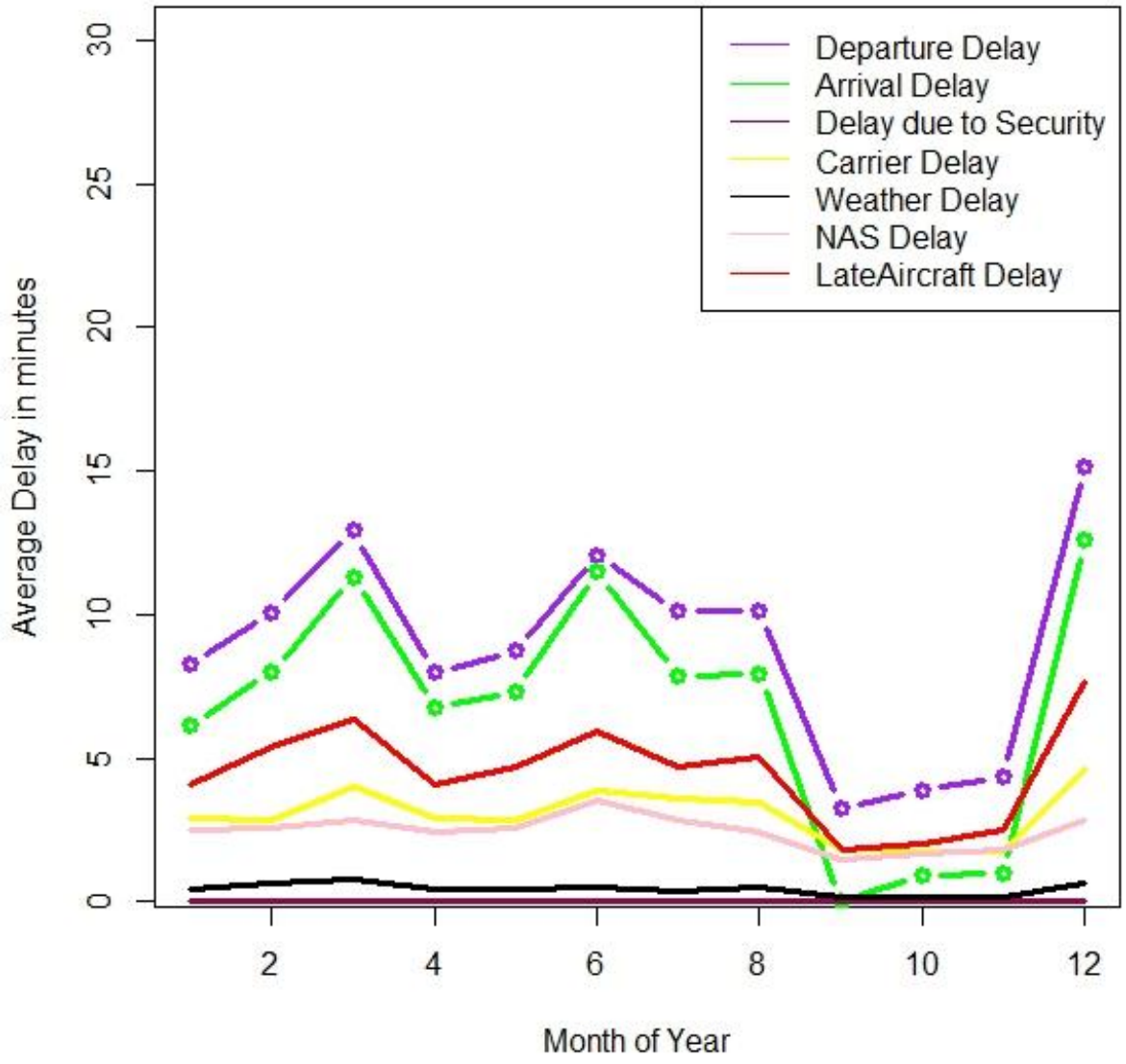


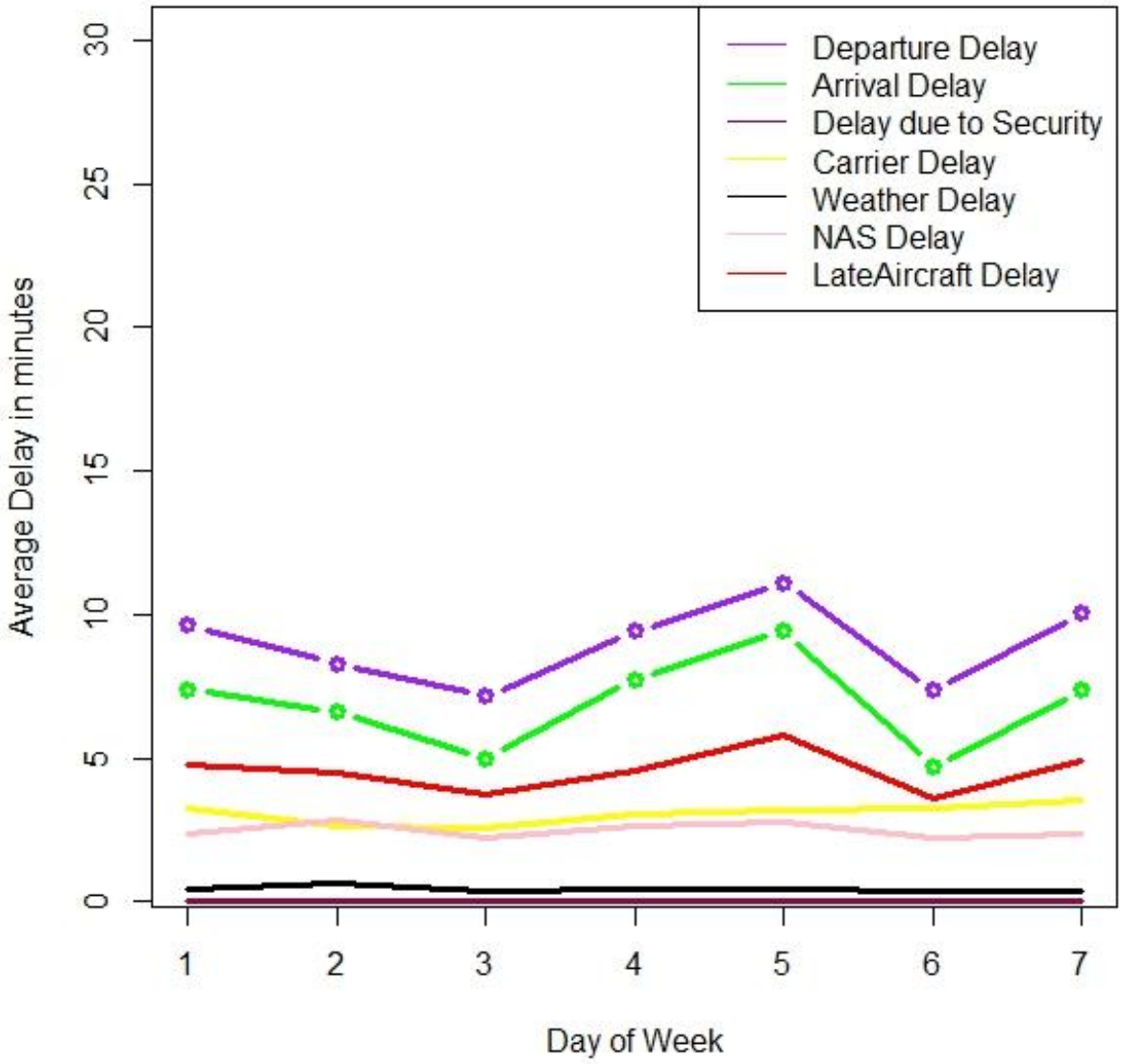
- It is clear that December is the worst month to fly and September is the best month to fly. One of the more probable explanation is : December is vacation time and one of the busiest time for airlines and airports. September is the begining of college/school season leading to lesser travel during this month. (observed that flight tickets are also cheapest during september-october and most expensive during Decmeber-Jan period)
- The cascading effect of the arrival delay onto the departure delay or vice-versa can be observed in the graph (they both go hand-in-hand)

**Similarly, plotting for weekly average delay below:**



- We can see that Fridays have the highest average delay followed by Sundays - the days that mark the begining and end of the weekend. These again are the busiest times of the week, because people are either flying out of their cities for the weekend or flying back in at the end of the weekend.
- **There is a more granular breakdown of delays in the dataset. Plotting similar line plots for these delays to analyse their effect on Arrival and Departure delay**






**Conclusion:**

* Can be observed that Weather delay and Security delay do not vary much across months/days of week and do not contribute much to the overall delay * The Late-Aircraft delay is the most dominant factor : Arrival delay at an airport due to the late arrival of the same aircraft at a previous airport. The ripple effect is significant

* Carrier delay is the next biggest contributing factor : Carrier delays can be due to aircraft cleaning, aircraft damage, awaiting the arrival of connecting passengers or crew, baggage, bird strike, cargo loading, catering.

* But carrier delay is something that the **aircraft carriers can control for**. Hence they should try and minimize this component when other delays which are not under their control are high

---

## Question 2 - Author Attribution

### Importing all the necessary libraries

### Defining a Reader plain function that wraps around the ReadPlain function and reads in the data

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

### Creating the Training Corpus

- Creating a file list for all 2500 docs being imported and assigning author names to each one of them

```
author_dirs_train = Sys.glob('C:/Users/Suraj/Desktop/James Scott/STA380-
master/data/ReutersC50/C50train/*')

file_list_train = NULL
train_labels = NULL

for(author in author_dirs_train) {
  author_name = substring(author, first=75)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list_train = append(file_list_train, files_to_add)
  train_labels = append(train_labels, rep(author_name, length(files_to_add)))
}
```

### Applying the ReaderPlain function for all the documents so that they will be read/imported in a certain manner

```
all_docs_train = lapply(file_list_train, readerPlain)
names(all_docs_train) = file_list_train
names(all_docs_train) = sub('.txt', '', names(all_docs_train))
```

### Creating a Corpus from the 'list' which has all the text from all the 'C50train' documents

```
train_corpus = Corpus(VectorSource(all_docs_train))
names(train_corpus) = file_list_train
```

### Performing certain pre-processing steps on the Training Corpus like:

- Removing numbers, punctuation(no emoticons here), whitespaces

- Convert to lower case
- Remove certain words, specified here under the 'SMART' kind of stopwords

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

### Creating a Document Term Matrix, where each row represents a document and each column represents a unique word from the entire corpus. The entries in this matrix are the counts for each of the words

```
DTM_train = DocumentTermMatrix(train_corpus)
DTM_train = removeSparseTerms(DTM_train, 0.96)
```

---

### Creating the Testing corpus

- Creating a file list for all 2500 docs being imported and assigning author names to each one of them

```
author_dirs_test = Sys.glob('C:/Users/Suraj/Desktop/James Scott/STA380-
master/data/ReutersC50/C50test/*')
file_list_test = NULL
test_labels = NULL
for(author in author_dirs_test) {
  author_name = substring(author, first=74)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list_test = append(file_list_test, files_to_add)
  test_labels = append(test_labels, rep(author_name, length(files_to_add)))
}
```

### Applying the ReaderPlain function for all the documents so that they will be read/imported in a certain manner

```
all_docs_test = lapply(file_list_test, readerPlain)
names(all_docs_test) = file_list_test
names(all_docs_test) = sub('.txt', '', names(all_docs_test))
```

### Creating a Corpus from the 'list' which has all the text from all the 'C50test' documents

```
test_corpus = Corpus(VectorSource(all_docs_test))
names(test_corpus) = file_list_test
```

### Performing certain pre-processing steps on the Training Corpus like:

- Removing numbers, punctuation(no emoticons here), whitespaces

- Convert to lower case
- Remove certain words, specified here under the 'SMART' kind of stopwords

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

### Standardizing the words in Test and Training dataset so that the test and train matrices match

- Creating a dictionary of words based on the training corpus
- Extracting these words from the test corpus

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

### Creating the testing DTM using words from the training dictionary

```
DTM_test = DocumentTermMatrix(test_corpus, list(dictionary=dict_train_words))
DTM_test = removeSparseTerms(DTM_test, 0.96)
```

### Converting DTMs to data frames

- Document Term matrices in their form do not work well for application of classifier models. Hence, will convert them to data frames
- The dataset is now in a format that can be used for classification models

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

---

## Naive Bayes model

- Running the Naive Bayes model to predict the authors of the docs in the Test dataset
- The naiveBayes function accounts for words not seen in training dataset through Laplace smoothing. (laplace = 1)

```
NB_Model = naiveBayes(x=DTM_train_df, y=as.factor(train_labels), laplace=1)
NB_prediction = predict(NB_Model, DTM_test_df)
```

- Creating a confusion matrix to calculate the accuracy of the model in predicting the authors
- Sensitivity column gives the accuracy % of predicting the documents under each of the authors correctly
- I have defined the accuracy of the model as the average of the accuracy measures for all the authors

```
CM_NB = confusionMatrix(table(NB_prediction,test_labels))
CM_NB_df = as.data.frame(CM_NB$byClass)
CM_NB_df[order(-CM_NB_df$Sensitivity),][1]
```

```
##                          Sensitivity
## Class: PeterHumphrey            0.90
## Class: RogerFillion             0.90
## Class: LydiaZajc                0.82
## Class: JimGilchrist             0.80
## Class: KouroshKarimkhany        0.78
## Class: AlanCrosby               0.68
## Class: LynneO'Donnell           0.64
## Class: RobinSidel               0.64
## Class: AaronPressman            0.54
## Class: BenjaminKangLim          0.50
## Class: TimFarrand               0.44
## Class: MatthewBunce             0.42
## Class: DavidLawder              0.40
## Class: TheresePoletti           0.40
## Class: JoWinterbottom           0.38
## Class: FumikoFujisaki           0.36
## Class: BernardHickey            0.28
## Class: JanLopatka               0.26
## Class: PierreTran               0.20
## Class: EricAuchard              0.18
## Class: GrahamEarnshaw           0.18
## Class: KarlPenhaul              0.18
## Class: SimonCowell              0.16
## Class: JonathanBirt             0.14
## Class: LynnleyBrowning          0.12
## Class: MarcelMichelson          0.12
## Class: NickLouth                0.12
## Class: JohnMastrini             0.10
## Class: KevinMorrison            0.10
## Class: HeatherScoffield         0.08
## Class: KeithWeir                0.08
## Class: KirstinRidley            0.08
## Class: MartinWolk               0.08
## Class: BradDorfman              0.06
## Class: JoeOrtiz                 0.06
## Class: MichaelConnor            0.06
## Class: AlexanderSmith           0.04
## Class: KevinDrawbaugh           0.04
## Class: PatriciaCommins          0.04
## Class: ToddNissen               0.04
## Class: SamuelPerry              0.02
## Class: TanEeLyn                 0.02
## Class: DarrenSchuettler         0.00
## Class: EdnaFernandes            0.00
## Class: JaneMacartney            0.00
```

```
## Class: MarkBendeich             0.00
## Class: MureDickie               0.00
## Class: SarahDavison             0.00
## Class: ScottHillis              0.00
## Class: WilliamKazer             0.00

Accuracy = mean(CM_NB_df$Sensitivity)
```

- Conclusion : The model has worked really well for a few authors like LydiaZajc, PeterHumphrey and RogerFillion. But for the majority of authors it hasn't been able to predict well. The accuracy % is low at around 25%.

---

## Random Forest model

### Converting Document term matrices into regular matrices, so that we can RandomForest model on it

- In it's current form however, Random Forest still cannot be applied because the number of columns in both the matrices is different

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

- One way to get around this problem is to append empty columns into the smaller matrix(test dataset in this case) before running Random Forest on them
- Below we are creating a dataframe from test dataset which has all the words occuring in the train dataset, but no words that occur in test but not in train
- For those words that did not occur in the 'C50Test' documents, the count will be 'NA'

```
com_data <- data.frame(DTM_test[,intersect(colnames(DTM_test),
colnames(DTM_train))])
eq_cols <- read.table(textConnection(""), col.names = colnames(DTM_train),
colClasses = "integer")
DTM_test_common = rbind.fill(com_data, eq_cols)
```

- We can now run RandomForest model using the dataset which is in the required format
- And then predict using the model

```
RF_model = randomForest(x=DTM_train_df, y=as.factor(train_labels), mtry=4,
ntree=200)
RF_predicted = predict(RF_model, data = DTM_test_common)
```

### Checking for model performance

```
RF_con_mat = confusionMatrix(table(RF_predicted,test_labels))

RF_con_mat$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.7088000      0.7028571      0.6905539      0.7265558      0.0200000
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

**Conclusion : The accuracy of the RF model comes out to be about 72%. This means that given a test document, the RF model can guess the right author 72 out of 100 times. This accuracy rate is much higher than that of the Naive Bayes model which had an accuracy of around 25%. Hence Random Forest does better than Naive Bayes for the Reuters example**

---

## Question 3 - Association Rule Mining

### Import the necessary libraries

### Importing dataset using 'read.transactions'. This function let's you import the dataset in the format which 'arules' can use

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

### Running the apriori algorithm on the dataset to generate association rules.

- Initially running 'apriori' with random values for the 'Support' and 'Confidence' parameters and checking the rules generated

```
groc_rules <- apriori(groceries, parameter=list(support=.01, confidence=.5, maxlen=6))
```

- The 15 rules generated here are the set of all possible association rules which have a support and confidence greater than the thresholds provided

```
inspect(groc_rules)
```

```
##    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
```

## Creating subsets of these association rules by altering the 'support', 'confidence' and 'lift' parameters and observing which association rules are filtered out

- 'Lift' is the increase in probability of the "consequent" itemset given the "if" (antecedent) itemset.
- Hence, higher the Lift, stronger is the association between the two itemsets in the association rule
- To filter out only the strong association rules we can subset for those rules which have high Lift
- In this example, no rules have a lift greater than 3.5

```
inspect(subset(groc_rules, 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
```

- We could get rules with a Lift greater than 3 but for that we will have to reduce the minimum 'Support' thresholds.
- This would give us rules where the association is stronger but, because 'Support' is low for them, the count of itemsets that show up in these rules are too low to be considered significant from a sales perspective.
- Similarly, getting high values of lift when 'Confidence' is low does not help, because this happens only when 'Expected Confidence' is also low. Such itemsets and the resultant association rules which have low 'Expected Confidence' may not be considered significant from a sales perspective.

**The highest values for 'Support' and 'Confidence' below which none of the rules show up are given below**

```
inspect(subset(groc_rules, subset=lift > 3))
inspect(subset(groc_rules, subset=confidence > 0.58))
inspect(subset(groc_rules, subset=support > .011 & confidence > 0.58))
```