Exercise 2

Mayur Srinivasan

18 August 2015

Q1

The ABIA dataset contains many variables that together explain the arrival and departure logistics at ABIA in 2008. Among them, the arrival and departure delays are one of the most important metrics whose analysis can yield many insights.

We attempt to map the average arrival and departure delays across various states that have flights to-and-from ABIA. We also look at the first and second half of 2008 separately to gain any additional seasonal insights.

We will first load a set of libraries that will help us plot shape files on R

```
library(rgeos)
library(maptools)
```

We then read in the shape file for US at a State level

```
np_dist <- readShapeSpatial("../data/USA_adm/USA_adm1.shp")
plot(np_dist)
```

```
library(ggplot2)

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

np_dist <- fortify(np_dist, region = "NAME_1")
```

We use another external file, "IATA_STATE" to use an IATA-to-State mapping. This will be used subsequently for mapping

```
IATA <- read.csv("../data/IATA_STATE.csv", header = TRUE)
AB <- read.csv("../data/ABIA.csv", header = TRUE)
```

We then merge these two datasets together to get the Arrival and Destination States for the corresposnding IATA codes (Airports)

```
total <- merge(AB,IATA[ , c("Origin", "State")],by="Origin", all.x = TRUE)
total <- merge(total,IATA[ , c("Dest", "State")],by="Dest", all.x = TRUE)
```

We then divide the data into 4 disjoint parta for analysis * Arrivals into ABIA in the first half (Months 1 to 6) of 2008 * Arrivals into ABIA in the second half of 2008 * Departures from ABIA in the first half (Months 1 to 6) of 2008 * Departures from ABIA in the second half of 2008

```
library(plyr)

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

total_new <- ddply(total[total$Month<=6 & total$Dest =='AUS',], .(State.x), s
ummarize, StateMean = mean(ArrDelay,na.rm = TRUE))
total_new1 <- ddply(total[total$Month>6 & total$Dest =='AUS',], .(State.x), s
ummarize, StateMean = mean(ArrDelay,na.rm = TRUE))
total_new2 <- ddply(total[total$Month<=6 & total$Origin =='AUS',], .(State.y)
, summarize, StateMean = mean(DepDelay,na.rm = TRUE))
total_new3 <- ddply(total[total$Month>6 & total$Origin =='AUS',], .(State.y),
summarize, StateMean = mean(DepDelay,na.rm = TRUE))
```

We are interested in analysing the mean delays (in minutes) in all 4 groups above

Below, we calculate the centres of the States to diplay the names of a few big states as a means of a complimentary legend

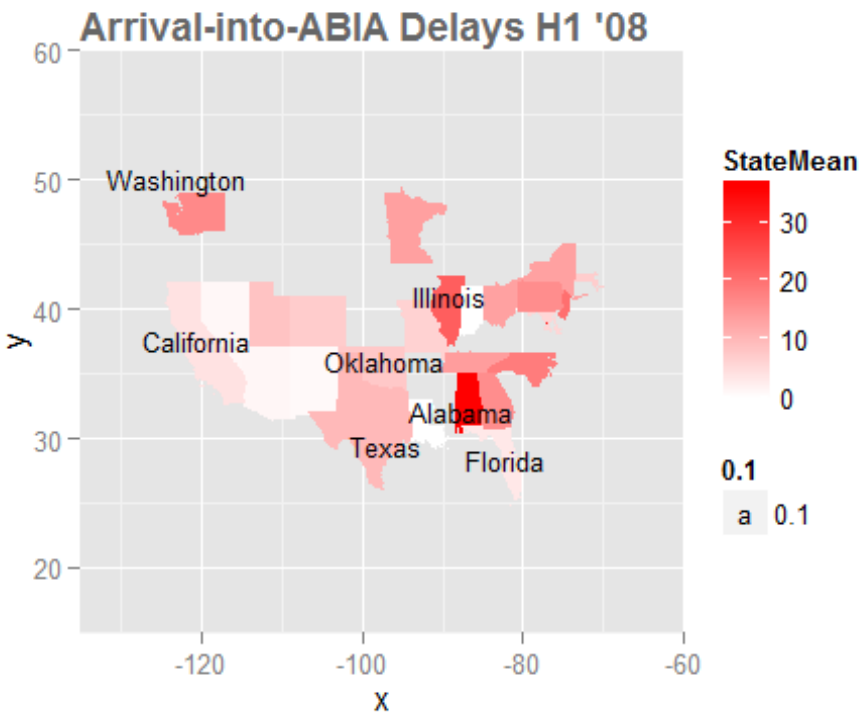
```
distcenters <- ddply(np_dist, .(id), summarize, clat = 1.05*mean(lat), clong
= mean(long))

distcenters <- subset(distcenters, id == 'Texas' | id == 'California'| id =='
Alabama'| id=='Oklahoma' | id =='Washington' | id == 'Illinois' | id == 'Flor
ida')
```

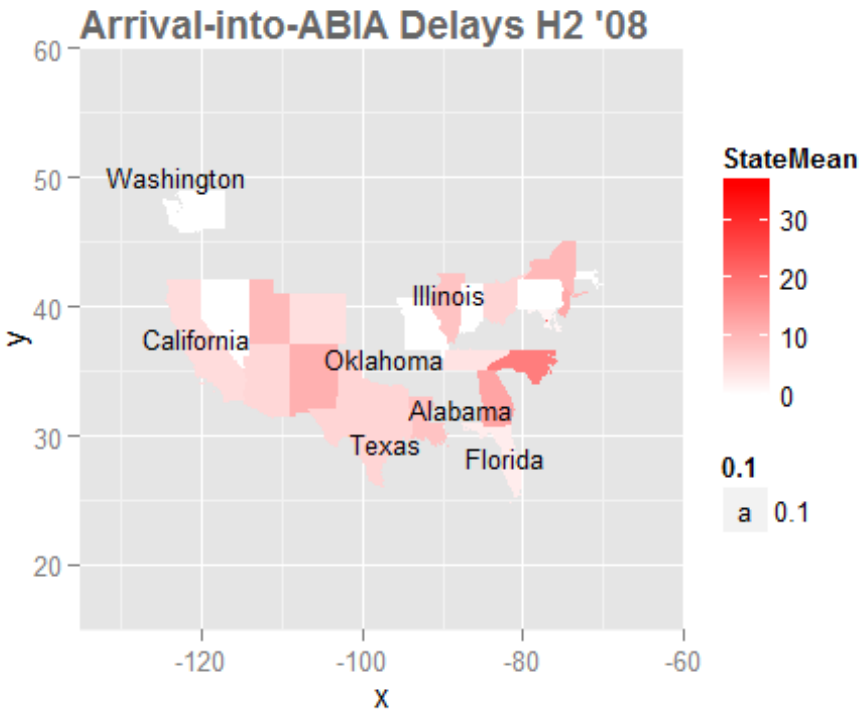
We first plot the Arrivals into ABIA in H1 '08

Only states from where there were arrivals into ABIA are plotted in the graph

```
ggplot() + geom_map(data = total_new, aes(map_id = State.x, fill = StateMean)
,
map = np_dist) + expand_limits(x = np_dist$long, y = np_dist$lat) +  coord_ca
rtesian(xlim = c(-135,-60),ylim = c(15,60)) +  scale_fill_gradient2(low = "wh
ite", midpoint = 0, high = "red", limits = c(-3, 36)) +
geom_text(data = distcenters, aes(x = clong, y = clat, label = id, size = 0.1
)) +
ggtitle("Arrival-into-ABIA Delays H1 '08") +
theme(plot.title = element_text(family = "Trebuchet MS", color="#666666", fac
e="bold", size=14, hjust=0))
```



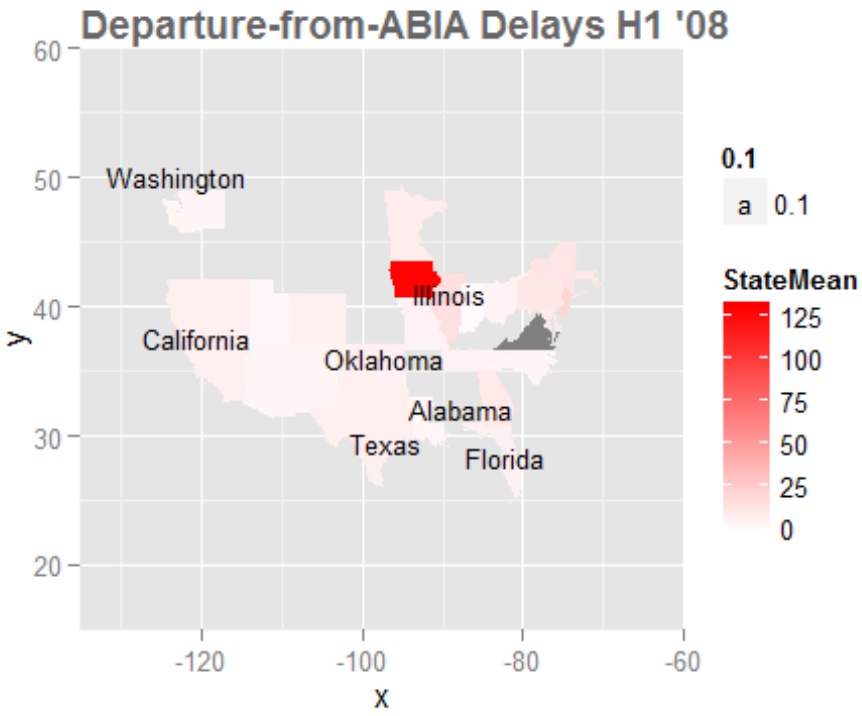

```
ggplot() + geom_map(data = total_new1, aes(map_id = State.x, fill = StateMean
),
map = np_dist) + expand_limits(x = np_dist$long, y = np_dist$lat) +  coord_ca
rtesian(xlim = c(-135,-60),ylim = c(15,60)) +  scale_fill_gradient2(low = "wh
ite", midpoint = 0, high = "red", limits = c(-3, 36)) +
geom_text(data = distcenters, aes(x = clong, y = clat, label = id, size = 0.1
)) +
ggtitle("Arrival-into-ABIA Delays H2 '08") +
theme(plot.title = element_text(family = "Trebuchet MS", color="#666666", fac
e="bold", size=14, hjust=0))
```



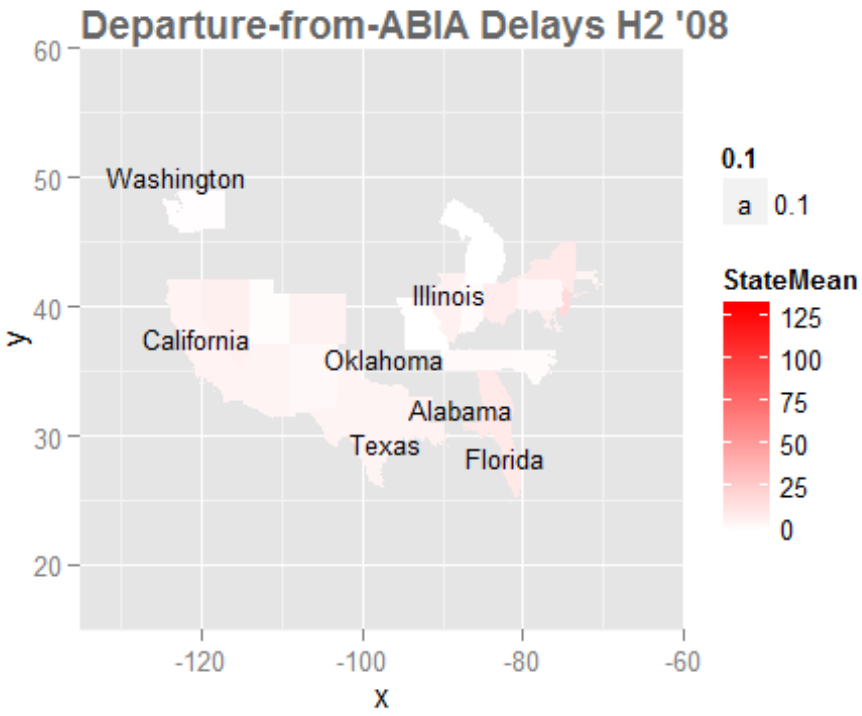
We see that the average delay in minutes for arrivals into ABIA were generally lower in the second half of 2008**. The North East registered longer arrival delays throughout the year with Illinois and Michigan leading the list of late arrivals**. Flights incoming from these states into ABIA can be expected to be late by around 30 minutes throughout the year, with the effect pronounced during the first half of the year.

We now plot the late departures from ABIA

```
ggplot() + geom_map(data = total_new2, aes(map_id = State.y, fill = StateMean
),
map = np_dist) + expand_limits(x = np_dist$long, y = np_dist$lat) +  coord_ca
rtesian(xlim = c(-135,-60),ylim = c(15,60)) +  scale_fill_gradient2(low = "wh
ite", midpoint = 0, high = "red", limits = c(0, 130)) +
geom_text(data = distcenters, aes(x = clong, y = clat, label = id, size = 0.1
)) +
ggtitle("Departure-from-ABIA Delays H1 '08") +
theme(plot.title = element_text(family = "Trebuchet MS", color="#666666", fac
e="bold", size=14, hjust=0))
```




```
ggplot() + geom_map(data = total_new3, aes(map_id = State.y, fill = StateMean
),
map = np_dist) + expand_limits(x = np_dist$long, y = np_dist$lat) +  coord_ca
rtesian(xlim = c(-135,-60),ylim = c(15,60)) +  scale_fill_gradient2(low = "wh
ite", midpoint = 0, high = "red", limits = c(0, 130)) +
geom_text(data = distcenters, aes(x = clong, y = clat, label = id, size = 0.1
)) +
ggtitle("Departure-from-ABIA Delays H2 '08") +
theme(plot.title = element_text(family = "Trebuchet MS", color="#666666", fac
e="bold", size=14, hjust=0))
```



Again, we observe that the length of delays in departures is on the higher side for the first half of 2008. Illinois is an outlier that skews the scale in H1 '08

Q2

We try 2 different models to predict the author of a set of documents while also trying to find any authors that demonstrate similar writing in terms of topics and subjects covered

Naive Bayes Model

We first load the tm library and execute the wrapper function to utilise the 'read files' utilities. This helps in reading and arranging all the input text files

```
library(tm)

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

We first read in all the filenames from the Train and Test folder separately. The filenames are then collected and appended in a vector that is subsequently used by the readerPlain function to produce a corpus

Please note that we take BOTH the Train and Test data to form the DocumentTermMatrix. This has been done to account for any new words/tokens that we might encounter in the Test data that is not part of the DTM of the Train data. It also ensures that the terms that are 'jointly sparse' i.e., Test terms that might be classified as sparse in comparison to Train data, are dealt with properly

```
author_dirs1 = Sys.glob('../data/ReutersC50/C50train/*')
author_dirs2 = Sys.glob('../data/ReutersC50/C50test/*')

file_list1 = NULL
file_list2 = NULL
labels = NULL
labels1 = NULL
labels2 = NULL

for(author in author_dirs1) {
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list1 = append(file_list1, files_to_add)
}

for(author in author_dirs2) {
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list2 = append(file_list2, files_to_add)
}

file_list3 = append(file_list1,file_list2)
```

Please note that the order in which the append happens is important because we want to ensure that the order of filenames and the subsequent corpus remains at the **"Test/Train X Author X Article"** level

We also extract the Author names from the directory names. This is useful to denote any author-level variables that we define later

```
for(author in author_dirs1) {
  author_name = substring(author, first=29)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  labels1 = append(labels1, rep(author_name, length(files_to_add)))
}

for(author in author_dirs2) {
  author_name = substring(author, first=28)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  labels2 = append(labels2, rep(author_name, length(files_to_add)))
}

labels <- unique(append(labels1, labels2))
```

Next, we extract the corpus using the readerPlain function and the mega-filelist that we compiled earlier. The corpus is alos put throuhg some basic cleaning using the utilities

provided within tm. This helps in addressing any redundancies and limits the noise-chasing that might occur with the subsequent model

```
all_docs = lapply(file_list3, readerPlain)
names(all_docs) = file_list3
names(all_docs) = sub('.txt', '', names(all_docs))

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

# Preprocessing
my_corpus = tm_map(my_corpus, content_transformer(tolower)) # make everything
lowercase
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers)) # remove nu
mbers
my_corpus = tm_map(my_corpus, content_transformer(removePunctuation)) # remov
e punctuation
my_corpus = tm_map(my_corpus, content_transformer(stripWhitespace)) ## remove
excess white-space
my_corpus = tm_map(my_corpus, content_transformer(removeWords), stopwords("SM
ART"))

DTM = DocumentTermMatrix(my_corpus)
DTM

## <<DocumentTermMatrix (documents: 5000, terms: 44234)>>
## Non-/sparse entries: 858721/220311279
## Sparsity           : 100%
## Maximal term length: 45
## Weighting          : term frequency (tf)
```

As expected, the sparsity of the Document Terms matrix is very high for any meaningful next step. We can use the inbuilt tm functionality of reduce the sparsity by a defined parameter

```
DTM = removeSparseTerms(DTM, 0.975)
DTM

## <<DocumentTermMatrix (documents: 5000, terms: 1386)>>
## Non-/sparse entries: 494509/6435491
## Sparsity           : 93%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

Next, the DTM is cnoverted to a data matrix, and the Train data is separated from the entire DTM. We then calculate the 'weight vector' for each Author after Laplace smoothing. These are the word/token level weights (for each author) that are multiplied with the word frequencies in the Test data to calculate the log probabilities eventually

The weight vector for each Author is named "**w_**"

```
X = as.matrix(DTM)

X_train <- X[1:2500,]
labels <- unique(labels)
smooth_count = 1/nrow(X_train)

for(i in 1:50)
{
  nam1 <- paste("w",labels[i], sep = "_")
  temp <- colSums(X_train[(50*i-49):(50*i),] + smooth_count)
  assign(nam1, temp/sum(temp))
}
```

We then 'predict' the author name on the Test data by calculating the log probabilities for each document across all authors and finding the highest value

```
X_test <- X[2501:5000,]

result = matrix(, nrow = 2500, ncol = 51)
for(i in 1:2500)
{ for(j in 1:50)
 {
  nam1 <- paste("w",labels[j], sep = "_")
  #check <- log(get(nam1))
  result[i,j] = sum(X_test[i,]*log(get(nam1)))
 }
}

result[1:5,1:10]
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## [1,]  -968.6253 -1205.9818 -1054.3074 -1213.7225 -1158.2985 -1136.5043
## [2,]  -591.0338  -741.1685  -681.2818  -670.1952  -762.9471  -689.4315
## [3,] -2185.4947 -2835.3524 -2494.5284 -2845.0260 -2532.8456 -2488.4620
## [4,]  -604.7419  -902.3168  -800.7661  -775.4366  -781.8511  -788.4264
## [5,] -1182.3740 -1663.7194 -1507.1808 -1531.8951 -1488.0430 -1461.1148
##            [,7]       [,8]       [,9]      [,10]
## [1,] -1110.1601 -1176.3673 -1119.1908 -1163.0840
## [2,]  -683.7230  -773.4523  -747.6904  -718.8658
## [3,] -2595.0482 -2846.5662 -2591.4150 -2200.6135
## [4,]  -790.6297  -856.0517  -793.5156  -803.1398
## [5,] -1532.5642 -1696.8449 -1547.7865 -1316.8529
```

We then append the predicted authors to the results dataset and then form a new dataset containing only the original author and the predicted author

```
for (i in 1:2500)
{
  result[i,51] = which.max(result[i,])
}
```

```
result1 = NULL
result1 = cbind((rep(1:50, each=50)),result[,51])
result1$auth <- rep(1:50, each=50)

## Warning in result1$auth <- rep(1:50, each = 50): Coercing LHS to a list

result1$pred_auth <- result[,51]
```

We then predict the accuracy of classification using the confusionMatrix utility of caret. This also provides the accuracy for each of the authors

```
library(caret)

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

## Loading required package: lattice
## Loading required package: ggplot2

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

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

confusionMatrix(result1$pred_auth,result1$auth)

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
##         1  42  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         2   0 25  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
##         3   0  0 20  0  2  0  0  0  3  0  0  0  0  0  0  0  3  5  0  2  0
##         4   0  0  0 11  0  0  0  0  0  0  0  0  0  0 10  0  0  0  0  0  0
##         5   0  0  0  0 27  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         6   1  0  0  0  0 43  0  8  0  1  0  0  0  0  0  0  0  0  0  0  0
##         7   1  0  0  0  0  0 14  0  0  0  0  0  7  0  0  0  0  0  0  0  0
##         8   0  0  0  0  0  0  0  7  0  0  0  0  0  0  0  0  0  0  0  0  0
##         9   0  0  0  0  0  0  0  0 20  0  0  0  0  0  0  0  3  0  0  1  0
##         10  0  0  0  0  0  2  0  0  0 25  0  0  0  0  0  0  0  0  0  0  0
##         11  0  0  0  0  0  0  0  0  0  0 49  0  0  0  0  0  0  0  0  0  0
##         12  0  0  0  2  0  0  0  0  0  0  0 39  0  0  2  0  0  0  0  0  0
##         13  0  0  0  0  3  0 36  0  0  0  0  0 17  0  0  0  0  0  0  0  0
##         14  0  8  0  0  0  0  0  0  0  0  0  0  1 26  0  0  0  0 11  0  0
##         15  0  0  0 13  0  0  0  1  0  0  0  3 12  0 18  0  0  0  0  0  0
##         16  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0 50  0  0  0  0  0
##         17  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0 33  0  0  2  0
##         18  1  0 28  0  1  0  0  1  0  0  0  0  0  0  0  0  1 39  0  0  0
##         19  0 17  0  0  0  0  0  0  0  0  0  0  3 23  0  0  0  0 32  0  0
```

```
##         20  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  2  0  0 37  0
##         21  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  2  0 47
##         22  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0
##         23  0  0  0  0  0  1  0  3  0  3  0  0  0  0  0  0  0  0  0  0  0
##         24  0  0  0  0  5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         25  0  0  0  0  0  0  0  0  5  2  0  0  0  0  0  0  0  0  0  1  0
##           Reference
## Prediction 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
##         1   0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  1
##         2   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         3   0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0
##         4   0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0
##         5   0  0  6  0  0  0  0  0  0  4  0  0  0  0  0  0  0  0  0  0  0
##         6   0  6  0  0  0  5  0  0  0  0  3  0  0  0  1  4  0  0  2  0  0
##         7   0  0  0  0  0  5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         8   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         9   5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
##         10  0  0  0  0 10  1  0  0  0  0  0  0  0  0  3  2  0  0  1  0  5
##         8   0  0  0  0  0  0 13  0
```

……

……

```
##         49  0  0  0  0  0  0 20  0
##         50  0  0  0  1  0  0  0 17
##
## Overall Statistics
##
##                Accuracy : 0.6024
##                  95% CI : (0.5829, 0.6217)
##     No Information Rate : 0.02
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5943
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.8400   0.5000   0.4000   0.2200   0.5400   0.8600
## Specificity            0.9963   0.9996   0.9869   0.9939   0.9955   0.9861
## Pos Pred Value         0.8235   0.9615   0.3846   0.4231   0.7105   0.5584
## Neg Pred Value         0.9967   0.9899   0.9877   0.9842   0.9907   0.9971
## Prevalence             0.0200   0.0200   0.0200   0.0200   0.0200   0.0200
## Detection Rate         0.0168   0.0100   0.0080   0.0044   0.0108   0.0172
## Detection Prevalence   0.0204   0.0104   0.0208   0.0104   0.0152   0.0308
## Balanced Accuracy      0.9182   0.7498   0.6935   0.6069   0.7678   0.9231
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity            0.2800   0.1400   0.4000    0.5000    0.9800
## Specificity            0.9947   0.9947   0.9951    0.9869    0.9988
## Pos Pred Value         0.5185   0.3500   0.6250    0.4386    0.9423
```

```
## Neg Pred Value         0.9854   0.9827   0.9878    0.9898    0.9996
## Prevalence             0.0200   0.0200   0.0200    0.0200    0.0200
## Detection Rate         0.0056   0.0028   0.0080    0.0100    0.0196
## Detection Prevalence   0.0108   0.0080   0.0128    0.0228    0.0208
## Balanced Accuracy      0.6373   0.5673   0.6976    0.7435    0.9894
##                      Class: 47 Class: 48 Class: 49 Class: 50
## Sensitivity             0.5200    0.7800    0.4000    0.3400
## Specificity             0.9902    0.9902    0.9849    0.9865
## Pos Pred Value          0.5200    0.6190    0.3509    0.3400
## Neg Pred Value          0.9902    0.9955    0.9877    0.9865
## Prevalence              0.0200    0.0200    0.0200    0.0200
## Detection Rate          0.0104    0.0156    0.0080    0.0068
## Detection Prevalence    0.0200    0.0252    0.0228    0.0200
## Balanced Accuracy       0.7551    0.8851    0.6924    0.6633
```

We can see that the overall accuracy rate of the Naive-Bayes classifier based on the articles is around 60% i.e., on an average 60% of the articles were correctly classified to their actual authors

We can also use these accuracy scores to find authors where the accuracy scores were low, and hence indicate that the other author that is most frequently predicted, has similar latent features according to the model and writes on similar subjects/topics

For example, from the confusion matrix we can observe that author 8 and author 49 have similar topics. A brief inspection does show that author 8 (David Lawder) and author 49 (Todd Nissen) write on the automobile industry with similar number of mentions of terms like 'Ford', 'auto', 'Detroit', 'car' etc.

We now test another classifier to see if we gain any Test accuracy. We first convert the data matrix to a data frame to be used in subsequent models

```
auth = rep(rep(1:50,each=50),2)
author = as.data.frame(X)
colnames(author) = make.names(colnames(author))
str(author)
```

```
## 'data.frame':    5000 obs. of  1386 variables:
##  $ ability           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ abroad            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ accept            : num  0 0 0 1 1 0 0 0 0 0 ...
##  $ access            : num  1 0 2 0 0 0 0 0 0 4 ...
##  $ account           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ accounting        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ accounts          : num  1 0 0 0 0 0 0 0 0 0 ...
##  $ accused           : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ achieve           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ acquire           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ acquired          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ acquisition       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ acquisitions      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ act               : num  0 0 0 0 0 0 1 5 1 0 ...
```

```
##  $ action            : num  0 0 0 0 0 0 0 1 0 0 ...
##  $ actions           : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ active            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ activities        : num  0 0 0 0 0 0 0 0 0 0 ...

##   [list output truncated]
```

```
author$auth=auth
author$auth=as.factor(author$auth)
```

The data is then divided into Train and Test as below:

```
author_train=author[1:2500,]
author_test=author[2501:5000,]
```

We then run a Random Forest model on the Train data and test is on the Test data

```
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.
```

```
set.seed(1)
authorRF=randomForest(auth~.,data=author_train)
preds=predict(authorRF,newdata=author_test)
confusionMatrix(preds,author_test$auth)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
##         1  46  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0  0  1  0  0  0
##         2   0 33  0  0  0  0  0  0  0  0  0  0  0  3  0  0  0  0  3  0  0
##         3   0  0 13  0  0  0  0  0  2  0  0  0  0  0  0  0  0  3  0  0  0
##         4   0  0  0 22  0  0  0  0  0  0  0  0  0  0 20  0  0  0  0  0  0
##         5   0  0  0  0 24  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         6   1  0  0  0  0 30  0  6  0  1  0  0  0  0  0  0  0  0  0  0  0
##         7   0  0  0  0  0  0 14  0  0  0  0  0 26  0  0  0  0  0  0  0  0
##         8   0  0  0  0  0  0  0  6  2  0  0  0  0  0  0  0  0  0  0  0  0
##         9   0  0  1  0  0  0  0  0 19  0  0  0  0  0  0  0  0  2  0  2  0
##         10  0  0  0  0  0  0  0  0  0 17  0  0  0  0  0  0  0  0  0  0  0
##         11  0  0  0  0  0  0  0  0  0  1 50  0  0  0  0  0  0  0  0  0  0
##         12  0  0  0  2  0  0  0  0  0  0  0 47  0  0  3  0  0  0  0  0  0
##         13  0  0  1  0  1  1 34  0  0  0  0  0 19  0  0  0  0  0  0  0  0
##         14  0  2  0  0  0  0  0  0  0  0  0  0  0 26  0  0  0  0 13  0  0
##         15  0  0  0  6  0  0  0  0  0  0  0  1  0  0 11  0  0  0  0  0  0
##         16  0  0  0  0  0  0  0  0  3  0  0  1  0  0  0 50  0  0  0  0  0
##         17  0  0  1  0  0  0  0  0  8  0  0  0  0  0  0  0 42  0  0  1  0
##         18  0  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  2 28  0  0  0
##         19  0 15  0  0  0  0  0  0  0  0  0  0  0 20  0  0  0  0 32  0  0
```

```
##         20  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  2  3  0 36  0
##         21  1  0  0  0  0  0  0  1  0  0  0  0  1  0  0  0  0  0  0  0 46
##         22  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         23  0  0  0  0  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         24  0  0  0  0 17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         25  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
##         26  0  0  0  0  0  0  0  0  0  6  0  0  0  0  0  0  0  0  0  0  0
##
```

……..

```
48  0  0  1  0  0 39  0  0
##         49  0  0  0  0  0  0 28  0
##         50  0  4  0  0  0  0  0 16
##
## Overall Statistics
##
##                Accuracy : 0.6196
##                  95% CI : (0.6002, 0.6387)
##     No Information Rate : 0.02
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6118
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.9200   0.6600   0.2600   0.4400   0.4800   0.6000
## Specificity            0.9927   0.9976   0.9971   0.9796   0.9947   0.9922
## Pos Pred Value         0.7188   0.8462   0.6500   0.3056   0.6486   0.6122
## Neg Pred Value         0.9984   0.9931   0.9851   0.9885   0.9894   0.9918
## Prevalence             0.0200   0.0200   0.0200   0.0200   0.0200   0.0200
## Detection Rate         0.0184   0.0132   0.0052   0.0088   0.0096   0.0120
## Detection Prevalence   0.0256   0.0156   0.0080   0.0288   0.0148   0.0196
## Balanced Accuracy      0.9563   0.8288   0.6286   0.7098   0.7373   0.7961
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity            0.2800   0.1200   0.3800    0.3400    1.0000
## Specificity            0.9816   0.9963   0.9951    0.9959    0.9963
## Pos Pred Value         0.2373   0.4000   0.6129    0.6296    0.8475
## Neg Pred Value         0.9853   0.9823   0.9874    0.9867    1.0000
## Prevalence             0.0200   0.0200   0.0200    0.0200    0.0200
## Detection Rate         0.0056   0.0024   0.0076    0.0068    0.0200
## Detection Prevalence   0.0236   0.0060   0.0124    0.0108    0.0236
## Balanced Accuracy      0.6308   0.5582   0.6876    0.6680    0.9982
##                      Class: 47 Class: 48 Class: 49 Class: 50
## Sensitivity             0.5600    0.7800    0.5600    0.3200
## Specificity             0.9922    0.9894    0.9800    0.9878
## Pos Pred Value          0.5957    0.6000    0.3636    0.3478
## Neg Pred Value          0.9910    0.9955    0.9909    0.9861
```

```
## Prevalence              0.0200    0.0200    0.0200    0.0200
## Detection Rate          0.0112    0.0156    0.0112    0.0064
## Detection Prevalence    0.0188    0.0260    0.0308    0.0184
## Balanced Accuracy       0.7761    0.8847    0.7700    0.6539
```

We see that the Random Forests model increases the accuracy of the classification by a miniscule amount, and talks about the same kind of associations as the Naïve Bayes (for example, author 8 and author 49)

Q3

We first load the arules library for the apriori calculation

```
library(arules)
```

We then read in the data at the transaction level, add a Transaction ID to identify them, as well as stack the data to bring it to a format that the subsequent split can understand

```
coln= max(count.fields("../data/groceries.txt",sep=','))
groc <- read.csv("../data/groceries.txt", header = FALSE,col.names = paste0("
V",seq_len(coln)),fill = TRUE)

groc$ID<-seq.int(nrow(groc))

groc[groc==''] <- 'Not Available'

out <- reshape(groc, direction = "long", idvar="ID",
               varying=(1:ncol(groc)-1), sep = "")


out <- out[order(out$ID),]
out$time <- NULL
out <- na.omit(out)


out$ID <- factor(out$ID)
```

We then split the data across each transaction and de-deuplicate the same to remove any similar transaction itemsets. The data is then cast into the 'transactions' class of arules

```
# First split data into a list of items for each transaction
out <- split(x=out$V, f=out$ID)

## Remove duplicates ("de-dupe")
out <- lapply(out, unique)

## Cast this variable as a special arules "transactions" class.
outtrans <- as(out, "transactions")


# Now run the 'apriori' algorithm
# Look at rules with support > .05 & confidence >.5
outrules <- apriori(outtrans, parameter=list(support=.05, confidence=.5, maxl
en=5))
```

```
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.5    0.1    1 none FALSE            TRUE    0.05      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 ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
# Look at the output
inspect(outrules)
```

```
## NULL
```

We can see that the thresholds for support and confidence used above are not enough to generate any significant rules. None of the itemsets generated above are well-represented at the given level and their 'consequents' are signigicant enough to follow from the 'antecedents'

We can reduce the minimum supoprt threshold to increase the base of itemsets considered

```r
# Look at rules with support > .01 & confidence >.5 & length (# items) <= 4
outrules1 <- apriori(outtrans, parameter=list(support=.01, confidence=.5, max
len=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 [0.03s].
## sorting and recoding items ... [88 item(s)] done [0.01s].
## creating transaction tree ... done [0.04s].
## checking subsets of size 1 2 3 4 done [0.02s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

```r
# Look at the output
inspect(outrules1)
```

```
##    lhs                     rhs                   support confidence     li
ft
## 1  {curd,
##     yogurt}             => {whole milk}       0.01006609  0.5823529 2.2791
25
## 2  {butter,
##     other vegetables}   => {whole milk}       0.01148958  0.5736041 2.2448
85
## 3  {domestic eggs,
##     other vegetables}   => {whole milk}       0.01230300  0.5525114 2.1623
36
## 4  {whipped/sour cream,
##     yogurt}             => {whole milk}       0.01087951  0.5245098 2.0527
47
## 5  {other vegetables,
##     whipped/sour cream} => {whole milk}       0.01464159  0.5070423 1.9843
85
## 6  {other vegetables,
##     pip fruit}          => {whole milk}       0.01352313  0.5175097 2.0253
51
## 7  {citrus fruit,
##     root vegetables}    => {other vegetables} 0.01037112  0.5862069 3.0296
08
```

```
## 8  {root vegetables,
##     tropical fruit}     => {other vegetables} 0.01230300  0.5845411 3.0209
99
## 9  {root vegetables,
##     tropical fruit}     => {whole milk}       0.01199797  0.5700483 2.2309
69
## 10 {tropical fruit,
##     yogurt}             => {whole milk}       0.01514997  0.5173611 2.0247
70
## 11 {root vegetables,
##     yogurt}             => {other vegetables} 0.01291307  0.5000000 2.5840
78
## 12 {root vegetables,
##     yogurt}             => {whole milk}       0.01453991  0.5629921 2.2033
54
## 13 {rolls/buns,
##     root vegetables}    => {other vegetables} 0.01220132  0.5020921 2.5948
90
## 14 {rolls/buns,
##     root vegetables}    => {whole milk}       0.01270971  0.5230126 2.0468
88
## 15 {other vegetables,
##     yogurt}             => {whole milk}       0.02226741  0.5128806 2.0072
35
```

We notice that the change in support generated some rules at the 0.5 level of confidence. We can inspect futher to find rules that have a higher level of significance (conditional probablity of rule being true given an itemset is true) OR have a higher lift (increase in the conditional probability of a 'consequent')

```
inspect(subset(outrules1, subset=confidence > 0.55))
```

```
##   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 {citrus fruit,
##    root vegetables}  => {other vegetables} 0.01037112  0.5862069 3.029608
## 5 {root vegetables,
##    tropical fruit}   => {other vegetables} 0.01230300  0.5845411 3.020999
## 6 {root vegetables,
##    tropical fruit}   => {whole milk}       0.01199797  0.5700483 2.230969
## 7 {root vegetables,
##    yogurt}           => {whole milk}       0.01453991  0.5629921 2.203354
```

```
inspect(subset(outrules1, subset=lift > 2))
```

```
##    lhs                     rhs                   support confidence     li
ft
## 1  {curd,
##     yogurt}             => {whole milk}       0.01006609  0.5823529 2.2791
25
## 2  {butter,
##     other vegetables}   => {whole milk}       0.01148958  0.5736041 2.2448
85
## 3  {domestic eggs,
##     other vegetables}   => {whole milk}       0.01230300  0.5525114 2.1623
36
## 4  {whipped/sour cream,
##     yogurt}             => {whole milk}       0.01087951  0.5245098 2.0527
47
## 5  {other vegetables,
##     pip fruit}          => {whole milk}       0.01352313  0.5175097 2.0253
51
## 6  {citrus fruit,
##     root vegetables}    => {other vegetables} 0.01037112  0.5862069 3.0296
08
## 7  {root vegetables,
##     tropical fruit}     => {other vegetables} 0.01230300  0.5845411 3.0209
99
## 8  {root vegetables,
##     tropical fruit}     => {whole milk}       0.01199797  0.5700483 2.2309
69
## 9  {tropical fruit,
##     yogurt}             => {whole milk}       0.01514997  0.5173611 2.0247
70
## 10 {root vegetables,
##     yogurt}             => {other vegetables} 0.01291307  0.5000000 2.5840
78
## 11 {root vegetables,
##     yogurt}             => {whole milk}       0.01453991  0.5629921 2.2033
54
## 12 {rolls/buns,
##     root vegetables}    => {other vegetables} 0.01220132  0.5020921 2.5948
90
## 13 {rolls/buns,
##     root vegetables}    => {whole milk}       0.01270971  0.5230126 2.0468
88
## 14 {other vegetables,
##     yogurt}             => {whole milk}       0.02226741  0.5128806 2.0072
35
```

We can make the following observations based on the rules that the above inspections give us:

- The {curd, yogurt} => {whole milk} rule has the highest life for milk and a very high confidence. This can be attributed to the natural practice of buying dairy items

together. This happens due to a combination of similar 'lifecycles' of dairy product consumption as well as the proximity placement of dairy items, compelling customers to buy generic dairy items at the same time

- Whole milk is a very common 'consequent' which indicates that any rule with Whole milk or products that usually associate well with Whole milk, are part of high confidence rules
- The ubiquity of basic groceries is apparent in the rules with vegetables as well. The category 'other vegetables' features quite frequently in 'antecedents' including 'root vegetables' and 'tropical fruit'