Assignment 2

Vijai Kasthuri Rangan

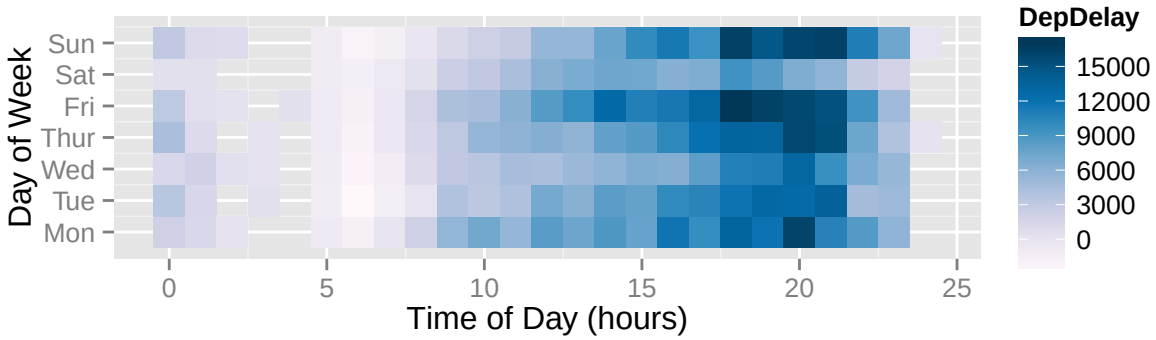
August 19, 2015

Ques 1. Flights at ABIA

1) What is the best time of day to fly to minimize delays?

To analyze the total delay in the departure of flights, a heat map is plotted that gives the aggregated delay across the different hours in a day for all the seven days of the week.

```
setwd("C:/Users/Vijai/OneDrive/Github/STA380 - 08132015/STA380-master/data")

flights = read.csv('ABIA.csv')

attach(flights)

flights$ArrTime_hour=as.integer(ArrTime/100)

flights$DepTime_hour=as.integer(DepTime/100)

departure_delay<- aggregate(DepDelay~DayOfWeek+DepTime_hour,flights,FUN='sum')

library(ggplot2)
library(RColorBrewer)

ggplot(departure_delay, aes(DepTime_hour,y=DayOfWeek))+
  geom_tile(aes(fill=DepDelay))+
  scale_fill_gradientn(colours=brewer.pal(9,"PuBu"),
                       breaks=seq(0,max(departure_delay$DepDelay),by=3000))+
  scale_y_continuous(breaks=7:1,labels=c("Sun","Sat","Fri","Thur","Wed","Tue","Mon"))+
  labs(x="Time of Day (hours)", y="Day of Week")+ coord_fixed()
```

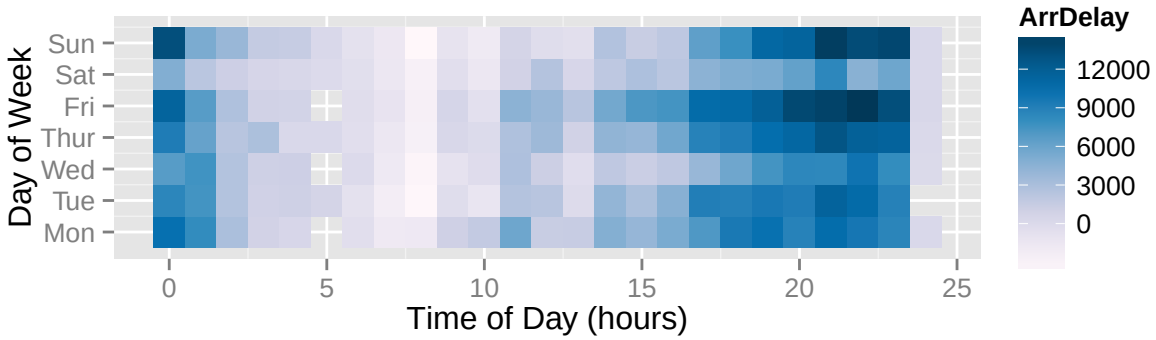


Observation :

From the above heatmap we find that there is a a lean period of no delays during the early hours of the day between 3 a.m - 7 a.m and hence this could be considered as the best time of the day to fly. We also observe that the intensity of the delay increases and peaks at 8:00 p.m in the evening. Th

The above graph only considers the departure delay to analyze the best time to fly. We could also check for the arrival delay to check for the best time of the day to fly.

```
Arrival_delay<- aggregate(ArrDelay~DayOfWeek+ArrTime_hour,flights,FUN='sum')
library(ggplot2)
library(RColorBrewer)
ggplot(Arrival_delay, aes(ArrTime_hour,y=DayOfWeek))+
  geom_tile(aes(fill=ArrDelay))+
  scale_fill_gradientn(colours=brewer.pal(9,"PuBu"),
                       breaks=seq(0,max(Arrival_delay$ArrDelay),by=3000))+
  scale_y_continuous(breaks=7:1,labels=c("Sun","Sat","Fri","Thur","Wed","Tue","Mon"))+
  labs(x="Time of Day (hours)", y="Day of Week")+ coord_fixed()
```

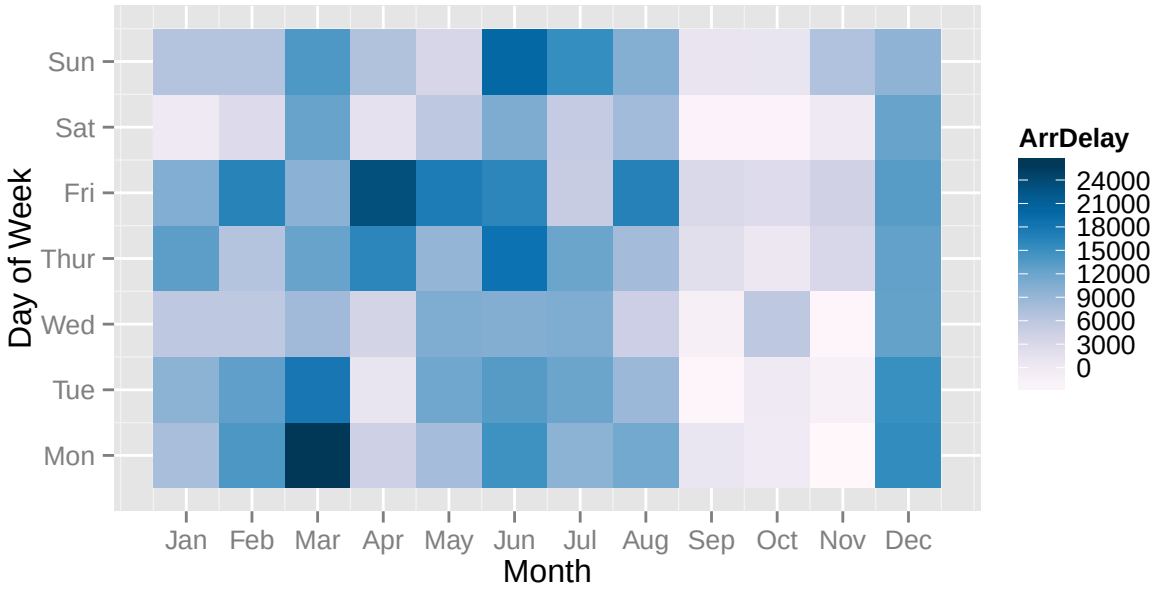


Observation :

The best time of the day to fly based on reduced arrival delay would be between 3 A.M and 10 A.M and also around 12 P.M and 2 P.M. The delay due to arrival increases during the later half of the day and peaks at evening 7 P.M to 11 P.M.

2) What is the best time of year to fly to minimize arrival delays?

```
arr_agg_month    <- aggregate(ArrDelay~DayOfWeek+Month,flights,FUN='sum')

ggplot(arr_agg_month, aes(Month,y=DayOfWeek))+
  geom_tile(aes(fill=ArrDelay))+
  scale_fill_gradientn(colours=brewer.pal(9,"PuBu"),
                       breaks=seq(0,max(arr_agg_month$ArrDelay),by=3000))+
  scale_y_continuous(breaks=7:1,labels=c("Sun","Sat","Fri","Thur","Wed","Tue","Mon"))+
  scale_x_continuous(breaks=1:12, labels=c("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct",
  labs(x="Month", y="Day of Week")+ coord_fixed()
```

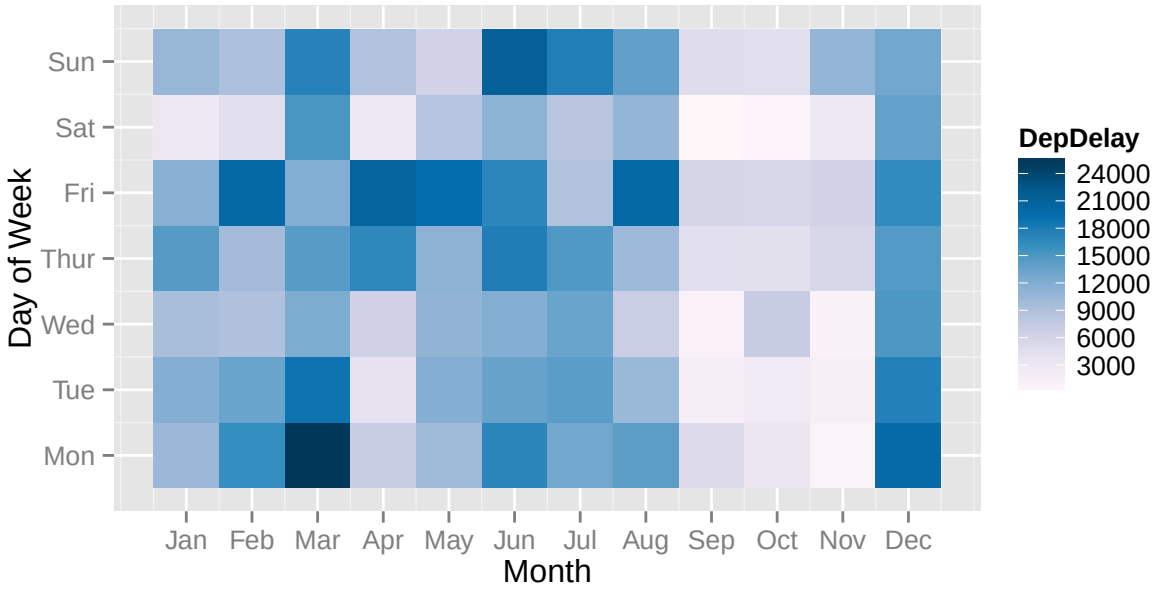


Observation :

The month of March June and December records the highest arrival delays in the year. from the heat map we find that The best time to fly in a year would be month of Sep, Oct, Nov and Apr. The month of April though has an unusual arrival delay on Fridays in comparison to the other days of the month which are relatively less delayed. Similarly Sundays on June experiences the highest delay.

2) What is the best time of year to fly to minimize departure delays?

```
Dep_agg_month<-aggregate(DepDelay~DayOfWeek+Month,flights,FUN='sum')

ggplot(Dep_agg_month, aes(Month,y=DayOfWeek))+
  geom_tile(aes(fill=DepDelay))+
  scale_fill_gradientn(colours=brewer.pal(9,"PuBu"),
                       breaks=seq(0,max(Dep_agg_month$DepDelay),by=3000))+
  scale_y_continuous(breaks=7:1,labels=c("Sun","Sat","Fri","Thur","Wed","Tue","Mon"))+ scale_x_continuou
  labs(x="Month", y="Day of Week")+ coord_fixed()
```



Observation :

The best time to fly during the year would be in the month of Sepetember, October and November based on the amount of departure delays. Even January records a relatively low levels of delays due to departure.

Q2 Author Attribution

The author attribution is accomplished by developing two separate models that use the document term matrices created on the different documents that have been written by the 50 different authors that we have in the training folder. Usng the models created on the train set we run the test set to check how the model performs.

Model1 - Naive Bayes

```
setwd("C:/Users/Vijai/OneDrive/Github/STA380 - 08132015/STA380-master/data")

#setting up the topic model library

library(tm)
```

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

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

Calling the readerplain function below to extract the contents of the documents under the training directory and loading it into all_docs

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

author_dirs = Sys.glob('../data/ReutersC50/C50train/*')
author_dirs = author_dirs[1:50]
file_list = NULL
labels = NULL
for(author in author_dirs) {
  author_name = substring(author, first=29)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  labels = append(labels, rep(author_name, length(files_to_add)))
}


# Need a more clever regex to get better names here
all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))
```

Creating a corpus based on the content retrieved from the training documents and renaming the corpus

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

Some additional preprocessing steps are added to enable the documents to make it usable for term frequency analysis and a document term matrix is created on top of that.

```
# Preprocessing
my_corpus = tm_map(my_corpus, content_transformer(tolower)) # make everything lowercase
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers)) # remove 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), stopwords("SMART"))

DTM = DocumentTermMatrix(my_corpus)
DTM # some basic summary statistics
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 31423)>>
## Non-/sparse entries: 425955/78131545
## Sparsity           : 99%
## Maximal term length: 36
## Weighting          : term frequency (tf)
```

```
DTM = removeSparseTerms(DTM, 0.945)

# Now a dense matrix
X = as.matrix(DTM)
```

On creating the required document term matrix for the training set we create the multinomial probability vectors for the different authors using the document term matrix created earlier. A smooth count is also added to the probability to account for less common words that we find in the training set.

```
w <- list()

smooth_count = 1/nrow(X)

for(i in 0:49) {

  w[[i+1]] = colSums(X[(i*50+1):((i+1)*50),] + smooth_count)
  w[[i+1]] = w[[i+1]]/sum(w[[i+1]])
}
```

A similar procedure to the training mentioned above is used to obtain the document term matrix for the test set of documents. There are 50 authors and each author has 50 different documents under his name in the test directory.

```
#Getting the X matrix for the test documents

author_dirs = Sys.glob('../data/ReutersC50/C50test/*')
author_dirs = author_dirs[1:50]
file_list = NULL
labels = NULL
for(author in author_dirs) {
  author_name = substring(author, first=27)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  labels = append(labels, rep(author_name, length(files_to_add)))
}


# Need a more clever regex to get better names here
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 everything lowercase
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers)) # remove 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), stopwords("SMART"))

DTM = DocumentTermMatrix(my_corpus)
class(DTM)  # a special kind of sparse matrix format
```

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

```
DTM = removeSparseTerms(DTM, 0.945)

# Now a dense matrix
X_test = as.matrix(DTM)

c <- as.array(NA, dim = c(50,1))
m <- matrix(data = NA, nrow = 2500, ncol = 2)


# Compare log probabilities under the Naive Bayes model - handles underflow and also makes the caluclat
```

The snippet below loops through the 2500 documents avaiable under the respective author names and then calculates the naive bayes log probability of these documents.

To handle the case of unseen words in the test document, a intersection of the columns between the test and training is used such that only those that are avialble in both the test and the training matrices are used to calculate the log probabilities

```
for( j in 1:2500)
{
  for( i in 1:50) {

    df_test <- t(data.frame(X_test[j,]))  #(X_test[j,])
    df_train <- data.frame(w[[i]]) #(w[[i]])
    df_train <- t(df_train)
    cmn <- intersect(colnames(df_train), colnames(df_test))
    df_train_updated <- df_train[,cmn]
    df_test_updated <- df_test[,cmn]
    X_test_mat <- as.matrix(df_test_updated)
    X_train_mat <- as.matrix(df_train_updated)

    c[i] = sum(X_test_mat * log(X_train_mat))
  }

  m[j,1] = max(c)
  m[j,2] = which.max(c)

}
```

We find the author names to use it to attribute the probabilites obtained from naive bayes to their corresponding author.

```
auth = substr(row.names(as.data.frame(X)), 29, length(row.names(as.data.frame(X))))
author = substr(auth, 1, regexpr("/",auth)-1)

m[,1] = author

author_new = unique(author)
```

Using the loop below we calculate the probability of finding the attributing the author to the one in the training set and comparing it with the authors listed in the test.

```
g <- matrix(data = NA, nrow=50, ncol=2)

for (i in 1:50)
{
  s <- table(m[((i-1)*50):(i*50),2])
  su <- s[names(s) == i]/50.0
  g[i,1] <- su
  g[i,2] <- author_new[i]
  #cat("probability of model to predict author", author_new[i], " correctly is ", su * 100, '\n' )
}


g <- as.data.frame(g)
```

The top 5 authors predicted by Naive Bayes:

```
head(g[order(g$V1, decreasing=TRUE),])
```

```
##      V1              V2
## 16 0.98    JimGilchrist
## 11 0.94  FumikoFujisaki
## 29 0.92 LynnleyBrowning
## 33 0.84    MatthewBunce
## 21 0.82     KarlPenhaul
## 28  0.8  LynneO'Donnell
```

The model performs the least on predicting the following authors

```
head(g[order(g$V1, decreasing=FALSE),])
```

```
##      V1               V2
## 8  0.12      DavidLawder
## 4   0.2  BenjaminKangLim
## 44 0.22      ScottHillis
## 7  0.24 DarrenSchuettler
## 9  0.24    EdnaFernandes
## 35 0.26       MureDickie
```

On an average Naive Bayes performs as given below:

```
cat("The Naive Bayes model on an average predicts ", sum(as.numeric(as.character(g[,1])))/50.0 * 100, "
```

```
## The Naive Bayes model on an average predicts  54.56 % accurately
```

Model2 - Random Forests

We create a random forest model based on the training documents available for the 50 authors in the training set. The random forest model created is then used to predict the authors of the testing documents. The

accuracy of predictions is calculated and the authors that can be easily identified and those that are difficult are obtained below.

Loading the required library. Using the test and the training documentts we create the DTM for these. The two DTM contain 2500 rows.

The DTMs are removed of any columns that are not available in either the test or the training DTM

```
###Model 2:
library(randomForest)
```

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

```
#Using the document term matrices created for test  & train
X_train <- X
X_test <- X_test

cmn_rf <- intersect(colnames(X_test), colnames(X_train))

X_train_trim <- X_train[,cmn_rf]
X_test_trim <- X_test[,cmn_rf]

df_X_train = as.data.frame(X_train_trim)
df_X_test = as.data.frame(X_test_trim)
```

Retrieving the respective author names and storing as the deoendent variable from the matrix (both test and train)

```
auth = substr(row.names(df_X_train), 29, length(row.names(df_X_train)))
author = substr(auth, 1, regexpr("/",auth)-1)
df_X_train["Author"] = author

authtest = substr(row.names(df_X_test), 28, length(row.names(df_X_test)))
authortest = substr(authtest, 1, regexpr("/",authtest)-1)

df_author = data.frame(unique(author))

row_index = 1:nrow(df_author)

df_X_test["Author"] = authortest

df_train_label = df_X_train[,"Author"]

df_X_train = df_X_train[,-531]

df_test_label = df_X_test[,"Author"]

df_X_test = df_X_test[,-531]
```

Running the random forest model on the training document term matrix and also predicting the model on the test data frame as below. The predicted values are then compared to the actual test values for the authors.

```
df_train.rf <- randomForest(x=df_X_train, y=as.factor(df_train_label), mtry=3, ntree=200)

predicted = predict(df_train.rf, df_X_test)

df = as.data.frame(predicted)

df["test"] = df_test_label

df["compare"] = as.numeric(df["test"] == df["predicted"])


df_predict = as.data.frame(unique(author))
for (i in 1:50)
{
    df_predict[i,"probab"] = sum(df[((i-1)*50):(i*50),"compare"])/50.0
}
```

The top 5 authors predicted by Random Forest:

```
#The top 5 authors predicted by Random Forest:

df_predict = df_predict[,c(2,1)]

head(df_predict[order(df_predict$probab, decreasing=TRUE),])
```

```
##    probab  unique(author)
## 16   1.00    JimGilchrist
## 41   0.92    RogerFillion
## 33   0.88    MatthewBunce
## 1    0.86   AaronPressman
## 21   0.86     KarlPenhaul
## 29   0.86 LynnleyBrowning
```

The model performs the least on predicting the following authors:

```
#The model performs the least on predicting the following authors

head(df_predict[order(df_predict$probab, decreasing=FALSE),])
```

```
##    probab unique(author)
## 44   0.08    ScottHillis
## 9    0.10  EdnaFernandes
## 15   0.14  JaneMacartney
## 49   0.18     ToddNissen
## 50   0.20   WilliamKazer
## 8    0.22    DavidLawder
```

From the above probability values we find that the Naive Bayes performs best on predicting the authors for the given test set.

```
cat("The probability of the random forest model is : ", sum(df["compare"])/2500 * 100, "%")
```

```
## The probability of the random forest model is :  51.24 %
```

From the comparisons of the two models we find that the Naive Bayes performs slightly better than the random forest on comparing the average probability of right predictions of the authors.

Q3 Association Rule Mining

To retrieve the associations among the different products purchased as a part of the different transaction we use the association rule mining. In order to use the association rule utilities we load the arules library below and add content of the transaction file to grocery variable. The read.transactions is used to retrieve the data as baskets that can be used for the arules command.

```
setwd("C:/Users/Vijai/OneDrive/Github/STA380 - 08132015/STA380-master/data")

options(warn = -1)

library(arules)

grocery <- read.transactions("groceries.txt",rm.duplicates = FALSE, format = "basket", sep = ",")

## Cast this variable as a special arules "transactions" class.

grocerytrans <- as(grocery, "transactions")
```

The apriori algorithm is used to identify the associations between the different products from the different baskets that were loaded. We choose the support value of 0.01 and a confidence of 0.1. This is the frequentist itemset generation process.

The support is the fraction of the transactions that contain an itemset. When we specify 0.01 for the support we wish to find association rules amongs the products that have a fraction of occurence among the different baskets to be >0.01.

The confidence measures how often the relation appears in the transactions containing the associated products in the basket.

```
# Now run the 'apriori' algorithm

groceryrules <- apriori(grocerytrans,
                        parameter=list(support=.01, confidence=.1, maxlen=6))
```

```
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.1    0.1    1 none FALSE            TRUE    0.01      1      6
##  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 [0.01s].
## checking subsets of size 1 2 3 4 done [0.01s].
## writing ... [435 rule(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

The subsets function below generate the required rules based on constraints on the lift, support and the confidence levels. The greater the level of confidence, greater is the chance that such a combination

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

From inspection we find that there are products that are being bought together and there are associations of these products.

Some interesting observations that we see are that when the condition on lift>3 we find an association of beef being bought along with root vegetables. Apart from this we also do get some intuitive observations like the basket of other vegetables bought along with orrt vegetables and citrus fruit.

```
## Choose a subset
inspect(subset(groceryrules, subset=lift > 3))
```

```
##   lhs                     rhs                      support confidence     lift
## 1 {beef}               => {root vegetables}    0.01738688  0.3313953 3.040367
## 2 {root vegetables}    => {beef}               0.01738688  0.1595149 3.040367
## 3 {whole milk,
##    yogurt}             => {curd}               0.01006609  0.1796733 3.372304
## 4 {other vegetables,
##    yogurt}             => {whipped/sour cream} 0.01016777  0.2341920 3.267062
## 5 {citrus fruit,
##    root vegetables}    => {other vegetables}   0.01037112  0.5862069 3.029608
## 6 {citrus fruit,
##    other vegetables}   => {root vegetables}    0.01037112  0.3591549 3.295045
## 7 {root vegetables,
##    tropical fruit}     => {other vegetables}   0.01230300  0.5845411 3.020999
## 8 {other vegetables,
##    tropical fruit}     => {root vegetables}    0.01230300  0.3427762 3.144780
```

For a subset with confidence > 0.53 we find whole milk being shopped in the basket along with other vegetables and citrus fruits! A support of 0.01 along with a confidence of 0.53 gives a similar result.

```
inspect(subset(groceryrules, subset=confidence > 0.53))
```

```
##   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(groceryrules, subset=support > .01 & confidence > 0.53))
```

```
##   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
```