# STA 380, Part 2: Exercises 2

*Mark Babbe, Camryn Callaway, Siqi Chen, Jiahao Ye, Zhiyi Yang*

*Aug 18, 2017*

## Contents

## Flights at ABIA

### Read in the data

```r
library(ggplot2)
library(gridExtra)
library(grid)
library(dplyr)
library(stringr)
library(plotly)

delay = read.csv('data/ABIA.csv')
attach(delay)
```

### Extract departure hour from departure time

```r
delay$CRSDepHr = as.factor(format(strptime(str_pad(delay$CRSDepTime,4,side = 'left',
                                                   pad = '0'),'%H%M'),'%H'))
```

### Separate the dataframe into departure and arrival

```r
departure = delay[delay$Origin == 'AUS',]
departure = departure[!is.na(departure$DepDelay),]
departure = departure[departure$DepDelay>0,]

arrival = delay[delay$Dest == 'AUS',]
arrival = arrival[!is.na(arrival$ArrDelay),]
arrival = arrival[arrival$ArrDelay>0,]
```

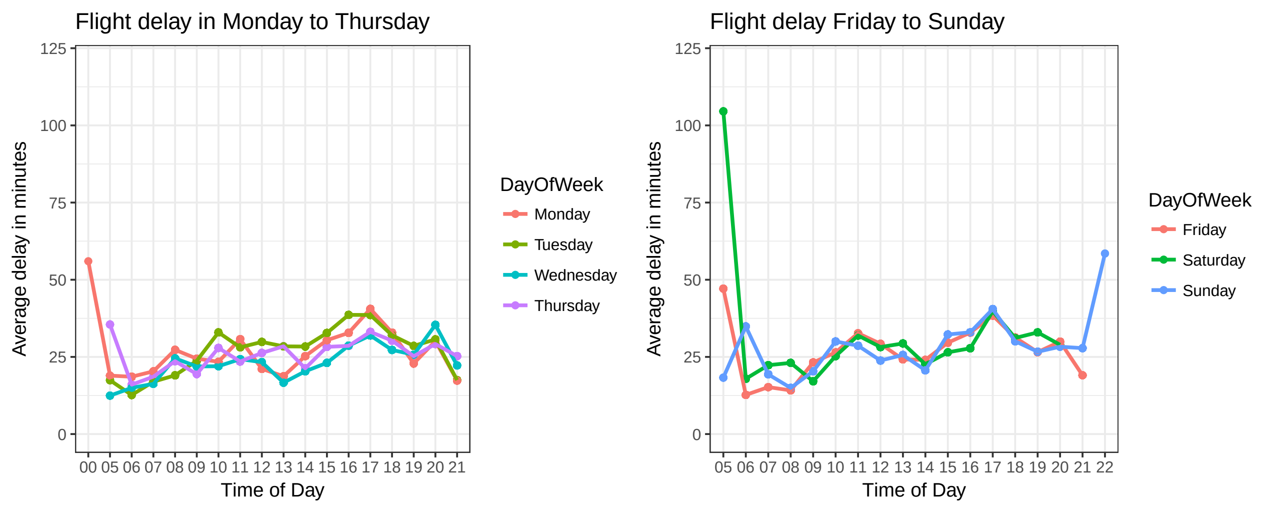
### Calculate delay average by hour, by day in a month and by destination

```r
departure$Month = as.factor(departure$Month)
departure$DayofMonth = as.factor(departure$DayofMonth)
departure$DayOfWeek = as.factor(departure$DayOfWeek)
departure$HrWkAvg = ave(departure$DepDelay,departure$DayOfWeek,departure$CRSDepHr)
```

```r
departure$DoMAvg = ave(departure$DepDelay,departure$Month,departure$DayofMonth)
departure$DestAvg = ave(departure$DepDelay,departure$Dest)

arrival$OrigAvg = ave(arrival$ArrDelay,arrival$Origin)
```

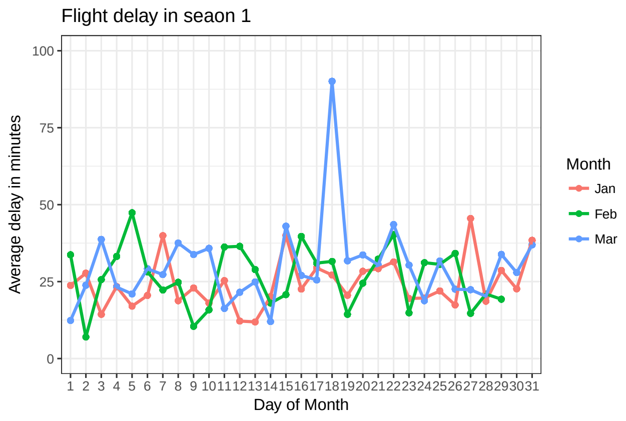
### Flight delay by hour of day in given weekday

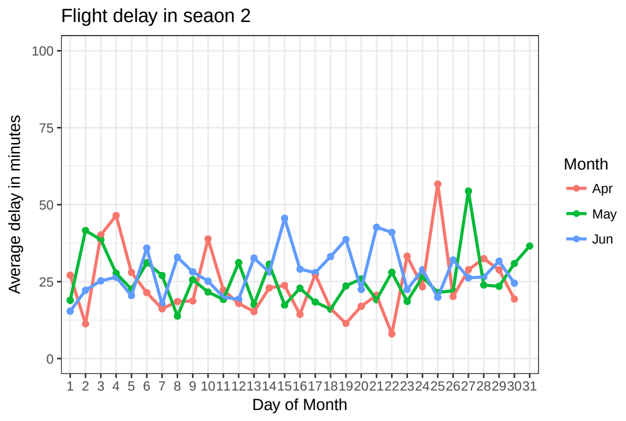
```r
MtoTH= subset(departure, DayOfWeek == 1|DayOfWeek==2|DayOfWeek==3|DayOfWeek==4)
FtoS = subset(departure, DayOfWeek == 5|DayOfWeek==6|DayOfWeek==7)

week1 = ggplot(MtoTH, aes(x = CRSDepHr, y = HrWkAvg, group = DayOfWeek, color = DayOfWeek)) +
  geom_point() +
  geom_line(size = 1) +
  ylim(0,120) +
  xlab('Time of Day') +
  ylab('Average delay in minutes') +
  ggtitle('Flight delay in Monday to Thursday') +
  scale_colour_discrete(labels=c("Monday","Tuesday","Wednesday","Thursday")) +
  theme_bw()

week2 = ggplot(FtoS, aes(x = CRSDepHr, y = HrWkAvg, group = DayOfWeek, color = DayOfWeek)) +
  geom_point() +
  geom_line(size = 1) +
  ylim(0,120) +
  xlab('Time of Day') +
  ylab('Average delay in minutes') +
  ggtitle('Flight delay Friday to Sunday') +
  scale_colour_discrete(labels=c("Friday","Saturday","Sunday"))+
  theme_bw()

grid.arrange(week1, week2, ncol = 2)
```

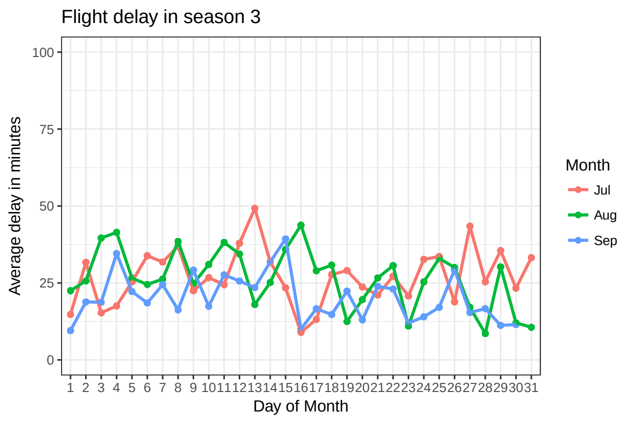


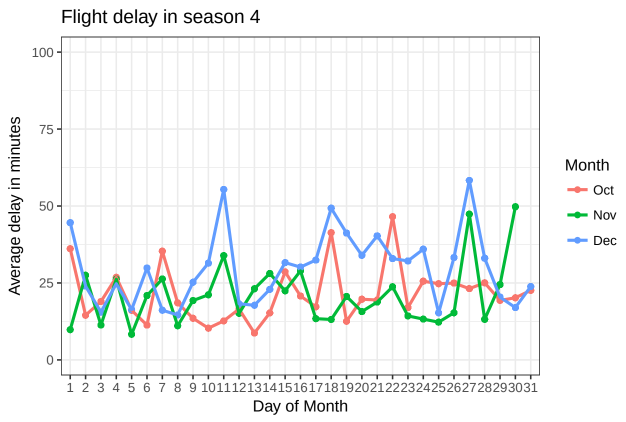
**Flight delay by day of month in given month**

```r
season1 = subset(departure, Month == 1|Month==2|Month==3)
season2 = subset(departure, Month == 4|Month==5|Month==6)
season3 = subset(departure, Month == 7|Month==8|Month==9)
season4 = subset(departure, Month == 10|Month==11|Month==12)

s1 = ggplot(season1, aes(x = DayofMonth, y = DoMAvg, group = Month, color = Month)) +
  geom_point() +
  geom_line(size = 1) +
  ylim(0,100) +
  xlab('Day of Month') +
  ylab('Average delay in minutes') +
  ggtitle('Flight delay in seaon 1') +
  scale_colour_discrete(labels=c("Jan","Feb","Mar"))+
  theme_bw()


s2 = ggplot(season2, aes(x = DayofMonth, y = DoMAvg, group = Month, color = Month)) +
  geom_point() +
  geom_line(size = 1) +
  ylim(0,100) +
  xlab('Day of Month') +
  ylab('Average delay in minutes') +
  ggtitle('Flight delay in seaon 2') +
  scale_colour_discrete(labels=c("Apr","May","Jun"))+
  theme_bw()


s3 = ggplot(season3, aes(x = DayofMonth, y = DoMAvg, group = Month, color = Month)) +
  geom_point() +
  geom_line(size = 1) +
  ylim(0,100) +
  xlab('Day of Month') +
  ylab('Average delay in minutes') +
  ggtitle('Flight delay in season 3') +
  scale_colour_discrete(labels=c("Jul","Aug","Sep"))+
  theme_bw()


s4 = ggplot(season4, aes(x = DayofMonth, y = DoMAvg, group = Month, color = Month)) +
  geom_point() +
  geom_line(size = 1) +
  ylim(0,100) +
  xlab('Day of Month') +
  ylab('Average delay in minutes') +
  ggtitle('Flight delay in season 4') +
  scale_colour_discrete(labels=c("Oct","Nov","Dec"))+
  theme_bw()

grid.arrange(s1,s2,s3,s4, nrow = 2, ncol = 2)
```









### Flight delay by destination and origin

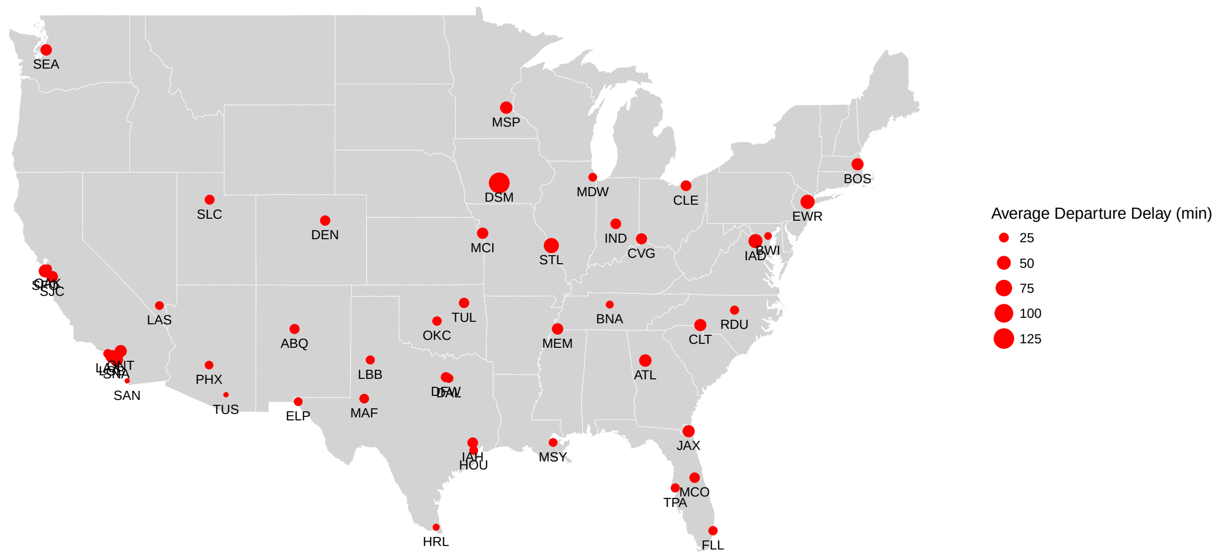
Longitude and latitude of the airports obtained online to plot them in a map

```
airport = read.csv('data/airport.csv')
departure = merge(departure,airport, by.x = 'Dest', by.y = 'airport')
arrival = merge(arrival, airport, by.x = 'Origin', by.y = 'airport')
usa = map_data("state")
```

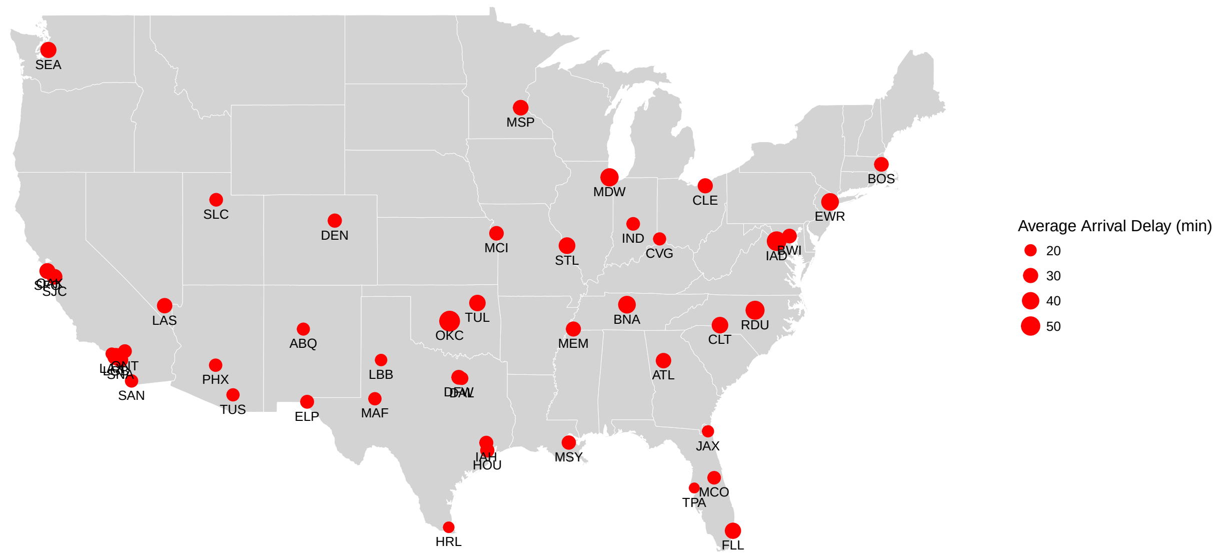
### Average departure and arrival delay

```
dest = departure[,c('Dest','DestAvg','lon','lat')]
dest = unique(dest)

plot_dest = ggplot() +
  geom_map(data=usa, map=usa,aes(x=long, y=lat, map_id=region),
           fill="lightgray", color="white", size=0.15) +
  geom_point(data = dest, aes(x = lon, y = lat, size = DestAvg), color = "red") +
  geom_text(data= dest, mapping=aes(x=lon, y=lat, label=Dest), size=3, vjust=2, hjust=0.5)+
  scale_size_continuous(name = "Average Departure Delay (min)") +
  ggtitle('2008 Austin Average Departure Delay by Destination') +
  theme_void()

orig = arrival[,c('Origin','OrigAvg','lon','lat')]
orig = unique(orig)
plot_orig = ggplot() +
  geom_map(data=usa, map=usa,aes(x=long, y=lat, map_id=region),
           fill="lightgray", color="white", size=0.15) +
  geom_point(data = orig, aes(x = lon, y = lat, size = OrigAvg), color = "red") +
  geom_text(data= orig, mapping=aes(x=lon, y=lat, label=Origin), size=3, vjust=2, hjust=0.5)+
  scale_size_continuous(name = "Average Arrival Delay (min)") +
  ggtitle('2008 Austin Average Arrival Delay by Origin') +
  theme_void()

grid.arrange(plot_dest,plot_orig, nrow = 2)
```

2008 Austin Average Departure Delay by Destination



2008 Austin Average Arrival Delay by Origin



## Author attribution

### Read in raw data from training and test sets

```
library(tm)
```

```
## Loading required package: NLP

##
## Attaching package: 'NLP'

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

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

file_list_train = Sys.glob('data/ReutersC50/C50train/*/*.txt')
file_list_test = Sys.glob('data/ReutersC50/C50test/*/*.txt')
all_train = lapply(file_list_train, readerPlain)
all_test = lapply(file_list_test, readerPlain)

names(all_train) = file_list_train
names(all_train) = substring(names(all_train),first=26)
names(all_train) = t(data.frame(strsplit(names(all_train),'/')))[,1]

names(all_test) = file_list_test
names(all_test) = substring(names(all_test),first=25)
names(all_test) = t(data.frame(strsplit(names(all_test),'/')))[,1]

my_documents_train = Corpus(VectorSource(all_train))
my_documents_test = Corpus(VectorSource(all_test))
```

### Process raw data

To make better prediction, we first make everything lowercase, then we remove numbers, puctuation and excess white-space. In addition, we also removed all the stopwords.

```
my_documents_train = tm_map(my_documents_train, content_transformer(tolower))
my_documents_train = tm_map(my_documents_train, content_transformer(removeNumbers))
my_documents_train = tm_map(my_documents_train, content_transformer(removePunctuation))
my_documents_train = tm_map(my_documents_train, content_transformer(stripWhitespace))

my_documents_train = tm_map(my_documents_train, content_transformer(removeWords), stopwords("en"))

my_documents_test = tm_map(my_documents_test, content_transformer(tolower))
my_documents_test = tm_map(my_documents_test, content_transformer(removeNumbers))
my_documents_test  = tm_map(my_documents_test, content_transformer(removePunctuation))
my_documents_test  = tm_map(my_documents_test, content_transformer(stripWhitespace))
my_documents_test  = tm_map(my_documents_test , content_transformer(removeWords), stopwords("en"))
```

### TF-IDF Score

Instead of purly using word count in each article, we decided to use TF-IDF score of each word within each article as the benchmark, since it adjust for relative frequency both in each article and among all articles, which can improve prediction accuracy. Moreover, we remove terms that are sparse more than 97.5%.

```
DTM_tfidf_train = DocumentTermMatrix(my_documents_train, control = list(weighting = weightTfIdf))
DTM_tfidf_train = removeSparseTerms(DTM_tfidf_train, 0.975)

DTM_tfidf_test = DocumentTermMatrix(my_documents_test, control = list(weighting = weightTfIdf))
DTM_tfidf_test = removeSparseTerms(DTM_tfidf_test, 0.975)
```

### Training and Test Sets

We then construct training and test sets from document term matrix. We added a column `author_train` and `author_test` to training and test sets respectively, as the response variables.

```
term_freq_train = as.data.frame(as.matrix(DTM_tfidf_train))
names(term_freq_train) = paste(names(term_freq_train),'.w',sep='')
author_train = factor(names(all_train))

term_freq_test = as.data.frame(as.matrix(DTM_tfidf_test))
names(term_freq_test) = paste(names(term_freq_test),'.w',sep='')
author_test = factor(names(all_test))

intersection = intersect(names(term_freq_train),names(term_freq_test))
term_freq_train = term_freq_train[,intersection]
term_freq_test = term_freq_test[,intersection]

X_train = term_freq_train
X_train$author = author_train
X_test = term_freq_test
X_test$author = author_test
```

### Model One: Naive Bayes

The first model we tried is Naive Bayes. The advantage of Naive Bayes is that it only requires simply calculation so the computation time is short. However, the average prediction accuracy among all authors is around 50%, which is not very satisfactory.

```
library(naivebayes)
nb.listing = naive_bayes(author ~ ., data = X_train)
nb.pred = data.frame(predict(nb.listing,X_test))
compare_nb = data.frame(cbind(nb.pred,X_test$author))
compare_nb$correct = compare_nb$predict.nb.listing..X_test. == compare_nb$X_test.author
mean(compare_nb$correct)
```

```
## [1] 0.502
```

### Model Two: Random Forest

The second model we tried is Random Forest. We expect tree model to give us higher prediciton accuracy. The average prediction accuracy among all authors from random forest is around 62.4%, which is a significant improvement comparing to Naive Bayes.

```
set.seed(10)
library(randomForest)
rf.listing = randomForest(author ~ ., data = X_train,
                          distribution = 'multinomial',
                          n.trees=200)
rf.pred = data.frame(predict(rf.listing,newdata = X_test))
compare_rf = data.frame(cbind(rf.pred,X_test$author))
compare_rf$correct = compare_rf$predict.rf.listing..newdata...X_test. == compare_rf$X_test.author
mean(compare_rf$correct)
```

```
## [1] 0.6244
```

### Result Comparison

To better understand the prediction result, we further calculate prediction accuracies for each author in both model.

```
library(pander)
authors = unique(names(all_train))
n_authors = length(authors)

correctness_nb = data.frame(matrix(ncol = 2, nrow = n_authors))
colnames(correctness_nb) = c('author', 'accuracy_nb')
correctness_nb$author = authors

for (i in 1:n_authors){
  set = subset(compare_nb,compare_nb$X_test.author == authors[i])
  correctness_nb[i,2] = mean(set$correct)
}

result_nb = correctness_nb[order(correctness_nb$accuracy_nb,decreasing = TRUE),]

correctness_rf = data.frame(matrix(ncol = 2, nrow = n_authors))
colnames(correctness_rf) = c('author', 'accuracy_rf')
correctness_rf$author = authors

for (i in 1:n_authors){
  set = subset(compare_rf,compare_rf$X_test.author == authors[i])
  correctness_rf[i,2] = mean(set$correct)
}

result_rf = correctness_rf[order(correctness_rf$accuracy_rf,decreasing = TRUE),]
```

### Naive Bayes

Top 10 accurate prediction

```
pander(result_nb[1:10,])
```

| | author | accuracy_nb |
|---|---|---|
| **29** | LynnleyBrowning | 0.88 |
| **33** | MatthewBunce | 0.86 |

|  | author | accuracy_nb |
|---|---|---|
| **11** | FumikoFujisaki | 0.8 |
| **1** | AaronPressman | 0.76 |
| **40** | RobinSidel | 0.74 |
| **25** | KirstinRidley | 0.72 |
| **6** | BradDorfman | 0.7 |
| **36** | NickLouth | 0.7 |
| **12** | GrahamEarnshaw | 0.68 |
| **28** | LynneO'Donnell | 0.68 |

Top 10 least accurate prediction

```
pander(result_nb[41:50,])
```

|  | author | accuracy_nb |
|---|---|---|
| **31** | MarkBendeich | 0.34 |
| **49** | ToddNissen | 0.32 |
| **9** | EdnaFernandes | 0.3 |
| **32** | MartinWolk | 0.3 |
| **15** | JaneMacartney | 0.28 |
| **3** | AlexanderSmith | 0.26 |
| **4** | BenjaminKangLim | 0.26 |
| **7** | DarrenSchuettler | 0.26 |
| **44** | ScottHillis | 0.16 |
| **8** | DavidLawder | 0.12 |

### Random Forest

Top 10 accurate prediction

```
pander(result_rf[1:10,])
```

|  | author | accuracy_rf |
|---|---|---|
| **11** | FumikoFujisaki | 1 |
| **16** | JimGilchrist | 1 |
| **29** | LynnleyBrowning | 0.98 |
| **1** | AaronPressman | 0.96 |
| **12** | GrahamEarnshaw | 0.96 |
| **21** | KarlPenhaul | 0.92 |
| **30** | MarcelMichelson | 0.92 |
| **33** | MatthewBunce | 0.92 |
| **38** | PeterHumphrey | 0.92 |
| **40** | RobinSidel | 0.92 |

Top 10 least accurate prediction

```
pander(result_rf[41:50,])
```

|  | author | accuracy_rf |
|---|---|---|
| **46** | TanEeLyn | 0.38 |
| **4** | BenjaminKangLim | 0.34 |

|  | author | accuracy_rf |
|---|---|---|
| **13** | HeatherScoffield | 0.34 |
| **50** | WilliamKazer | 0.34 |
| **32** | MartinWolk | 0.32 |
| **9** | EdnaFernandes | 0.3 |
| **7** | DarrenSchuettler | 0.28 |
| **10** | EricAuchard | 0.24 |
| **8** | DavidLawder | 0.14 |
| **44** | ScottHillis | 0.14 |

Overall, random forest has much better prediction accuracy for top accurate predictions. In particular, prediction accuracies for articles from *Fumiko Fujisaki* and *Jim Gilchrist* are 100%. However, for least accurate predicitions, both models yield similar predicion accuracy. Articles from *David Lawder* and *Scott Hillis* are hardest to predict. Both models have less than 20% prediction accuracies.

## Practice with association rule mining

### Read in raw file as transactions

```
library(arules)
```

```
## Loading required package: Matrix

##
## Attaching package: 'arules'

## The following object is masked from 'package:tm':
##
##     inspect

## The following object is masked from 'package:dplyr':
##
##     recode

## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```
groceries = read.transactions(file = 'data/groceries.txt', rm.duplicates = TRUE,
                              format = 'basket', sep = ',')
```

### Apply Apriori algorithm

After trying several combinations of min support and min confidence, we set the initial threshold for support at 0.015, and confidence at 0.25, with a maximum of 3 items in itemlist. There are around 10000 transactions, so an itemlist need to at least appears 150 times base on 0.015 minimum support, which is a reasonable amount. If the min support is too high, the resulting associations have low lift and are not very meaningful. On the other hand, if the min support is too low, the resulting associations have high lift, but might not be very representative.

```
grocery_rules = apriori(groceries,
                        parameter=list(support=.015, confidence=.25, maxlen=3))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.25    0.1    1 none FALSE            TRUE       5   0.015      1
##  maxlen target   ext
##       3  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 147
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [73 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3
```

```
## Warning in apriori(groceries, parameter = list(support = 0.015, confidence
## = 0.25, : Mining stopped (maxlen reached). Only patterns up to a length of
## 3 returned!
```

```
##  done [0.00s].
## writing ... [79 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

With the above threshold, we have 79 resulting associations, so we decided to further cutdown the list.

### Result inspection

We first look at the ones with lift higher than 2.5, which means the conditional probability of purchasing is at least 2.5 times higher than the marginal probability. The results are as follows:

```r
inspect(subset(grocery_rules, subset=lift > 2.5))
```

```
##     lhs                                rhs                support
## [1] {beef}                          => {root vegetables} 0.01738688
## [2] {pip fruit}                     => {tropical fruit}  0.02043721
## [3] {tropical fruit,whole milk}     => {yogurt}          0.01514997
## [4] {whole milk,yogurt}             => {tropical fruit}  0.01514997
## [5] {other vegetables,whole milk} => {root vegetables} 0.02318251
##     confidence lift
## [1] 0.3313953  3.040367
## [2] 0.2701613  2.574648
## [3] 0.3581731  2.567516
## [4] 0.2704174  2.577089
## [5] 0.3097826  2.842082
```

All five results make sense to us. People tend to buy carrot, potaoes with beef; buy different kinds of fruits together; buy tropical fruit with milk and yogurt for smoothies and milkshakes; and buy vegetables with milk maybe for some 'healthy' drink.

We then look at the ones with confidence higher than 0.5, which means the subset appears at least half of the times in the itemlist. The results are as follows:

```r
inspect(subset(grocery_rules, subset=confidence > 0.5))
```

```
##     lhs                             rhs          support    confidence
## [1] {tropical fruit,yogurt}      => {whole milk} 0.01514997 0.5173611
## [2] {other vegetables,yogurt} => {whole milk} 0.02226741 0.5128806
##     lift
## [1] 2.024770
## [2] 2.007235
```

These two results are similar to the some from previous inspection, the combination of fruit, yogurt and milk seem to be good ingredients for milkshake and smoothies.

Lastly, we look at the ones with slightly lower confidence of 0.3, but higher support of 0.05.

```r
inspect(subset(grocery_rules, subset=support > .05 & confidence > 0.3))
```

```
##     lhs                   rhs          support    confidence lift
## [1] {yogurt}           => {whole milk} 0.05602440 0.4016035  1.571735
## [2] {rolls/buns}       => {whole milk} 0.05663447 0.3079049  1.205032
## [3] {other vegetables} => {whole milk} 0.07483477 0.3867578  1.513634
```

These three associations also make sense, though the lifts are not as high as the previous associations. First two can be associations for breakfast, the third one might be for ‘healthy’ food style people.