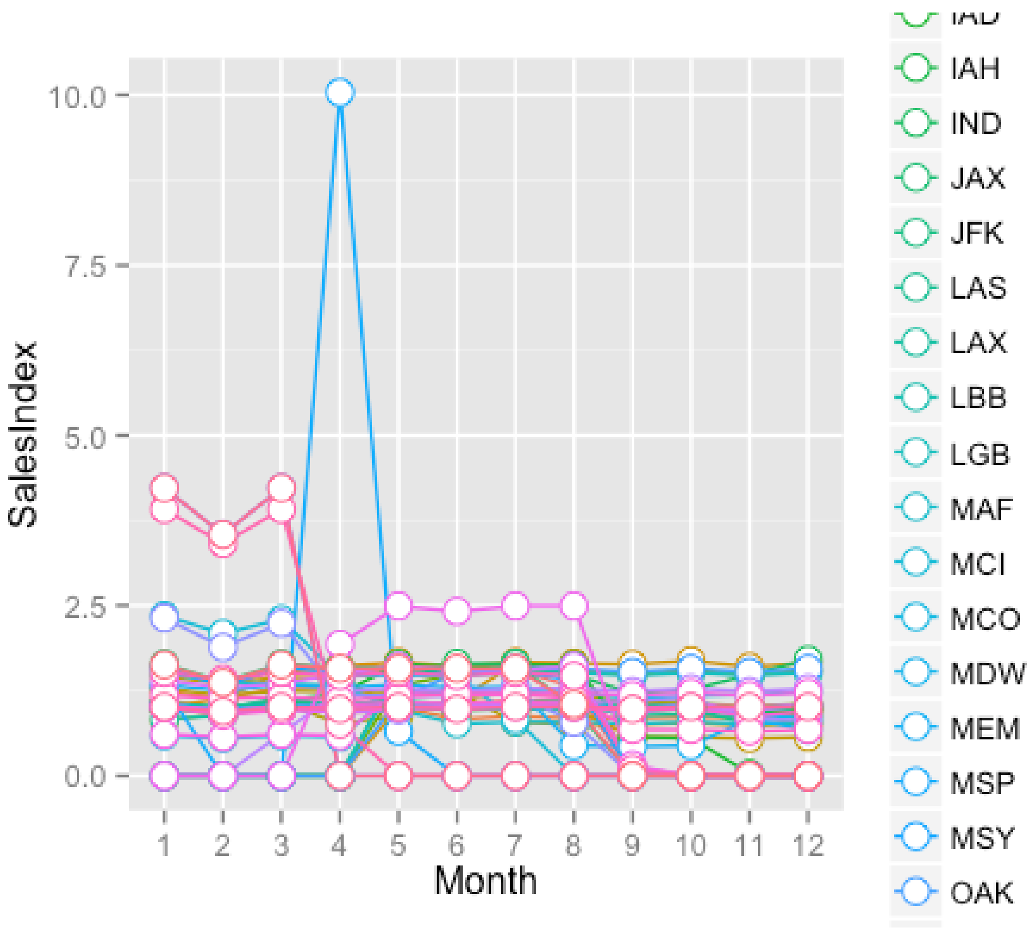
Homework 2

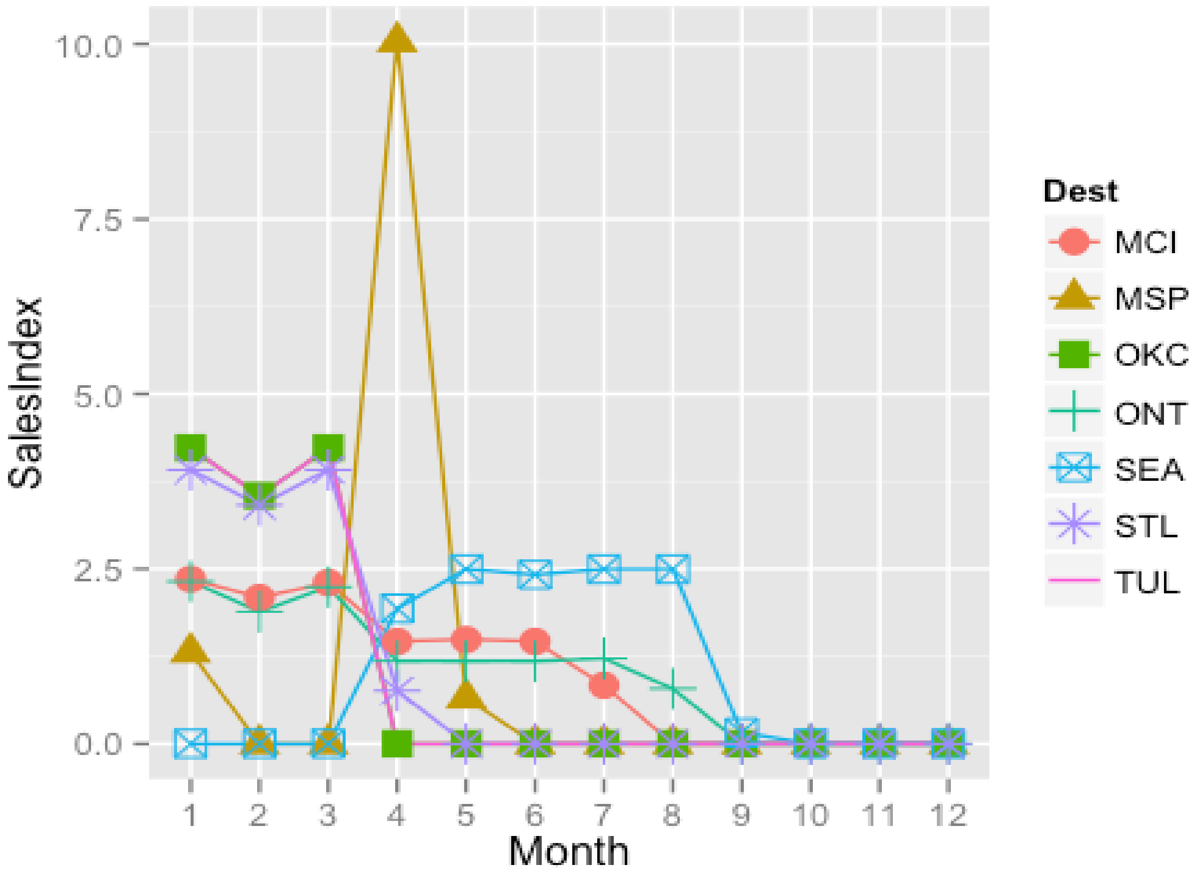
Michael Zhang

Flights at ABIA

```r
library(ggplot2)
abia <- read.csv('../data/ABIA.csv')

#select only outbound flights from AUS
outbound <- subset(abia, abia$Origin == 'AUS' & abia$Dest != 'AUS')

#crosstab of monthly flights
mnthlyflights = xtabs(~ Month + Dest, outbound)

#create data frame from crosstab
df <- as.data.frame(mnthlyflights)

#create vectors of months and airports
months <- unique(outbound$Month)
airports <- unique(outbound$Dest)

#calculate average # of monthly flights to each destination city
df$meanfreq <- ave(df$Freq, df$Dest)

#remove any cities with <= 1 flight/month
dfclean <- subset(df, df$meanfreq > 1)

#calculate sales index (indexing each month's # of flights by that city's
monthly average for the year)
dfclean$SalesIndex <- dfclean$Freq/dfclean$meanfreq

#plot showing all monthly outbound flight trends
ggplot(data=dfclean, aes(x=Month, y=SalesIndex, group = Dest, colour = Dest))
+ geom_line() + geom_point(size=4, shape=21, fill="white")
```



```
#create column with max SIs for each city
maxSI <- aggregate(SalesIndex ~ Dest, dfclean, max)
colnames(maxSI) <- c("Dest", "SImax")
dfmerged <- merge(dfclean, maxSI, by = "Dest")

#remove any cities with a max SI <= 2
citiesofinterest <- subset(dfmerged, SImax > 2)
ggplot(data=citiesofinterest, aes(x=Month, y=SalesIndex, group = Dest, colour
= Dest)) + geom_line() + geom_point(size=4, aes(shape=Dest))
```



"How do patterns of flights from Austin to different destinations change over the course of the year?"

To answer this question, I looked at the seasonality of outbound flights from AUS in 2008. First, I used each destination's monthly average number of flights from AUS as the base index to calculate the monthly sales indices for each destination.

For example, the average number of monthly flights from Austin (AUS) to Kansas City (MCI) in 2008 was 38.25. During Jan-Mar, the sales indices for MCI were above 2, meaning that the flights during those months were more than twice the average number of monthly flights (> 2 * 38.25).

The plot above shows 7 destinations of particular interest. Specifically, these cities were found to have the largest seasonal variation. A few observations:

- The majority of destinations had very consistent levels of outbound flights from AUS on a monthly basis.
- Several destinations (Tulsa, Oklahoma City, St. Louis, Kansas City, and LA/Ontario) overindex in flights during the first quarter of the year.

- While Kansas City and LA/Ontario maintain slightly lower flight levels through July/August, the rest of these destinations completely drop off in terms of flight activity from Austin.
- Seattle is primarily a spring/summer destination for Austinites.
- In April, there were 46 flights from AUS to MSP. The only two other months with flights to MSP were January and May, which had 6 and 3 flights, respectively.

Intuitively, it's easy to make sense of the observation about Seattle--people are more likely to visit when the weather is nicer there. However, it's hard to make sense of the other observations without other knowledge about events in those cities or work travel patterns.

Author Attribution

```
library(tm)
library(e1071)
library(RTextTools)

# Remember to source in the "reader" wrapper function
readerPlain = function(fname) {
  readPlain(elem=list(content=readLines(fname)), id=fname, language = 'en')
}

## Rolling 50 directories together into a single corpus
train_author_dirs = Sys.glob('../data/ReutersC50/C50train/*')
train_file_list = NULL
train_labels = NULL
for(author in train_author_dirs) {
      author_name = substring(author, first=29)
      files_to_add = Sys.glob(paste0(author, '/*.txt'))
      train_file_list = append(train_file_list, files_to_add)
      train_labels = append(train_labels, rep(author_name, 
length(files_to_add)))
}
```

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

# Create training corpus
training_corpus = Corpus(VectorSource(all_docs))
names(training_corpus) = names(all_docs)

# Preprocessing
training_corpus = tm_map(training_corpus, content_transformer(tolower))
training_corpus = tm_map(training_corpus, content_transformer(removeNumbers))
training_corpus = tm_map(training_corpus,
content_transformer(removePunctuation))
training_corpus = tm_map(training_corpus,
content_transformer(stripWhitespace))
training_corpus = tm_map(training_corpus, content_transformer(removeWords),
stopwords("SMART"))

# Create a document-term matrix and remove sparse terms
training_DTM = DocumentTermMatrix(training_corpus)
training_DTM = removeSparseTerms(training_DTM, 0.96)
train_DTM = as.matrix(training_DTM)

# Do same thing for test set
test_author_dirs = Sys.glob('../data/ReutersC50/C50test/*')
test_file_list = NULL
test_labels = NULL
for(author in test_author_dirs) {
      author_name = substring(author, first=28)
      files_to_add = Sys.glob(paste0(author, '/*.txt'))
      test_file_list = append(test_file_list, files_to_add)
      test_labels = append(test_labels, rep(author_name,
length(files_to_add)))
}


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


# Create testing corpus
testing_corpus = Corpus(VectorSource(all_docs))
names(testing_corpus) = names(all_docs)


# Preprocessing
testing_corpus = tm_map(testing_corpus, content_transformer(tolower))
testing_corpus = tm_map(testing_corpus, content_transformer(removeNumbers))
testing_corpus = tm_map(testing_corpus,
content_transformer(removePunctuation))
testing_corpus = tm_map(testing_corpus, content_transformer(stripWhitespace))
testing_corpus = tm_map(testing_corpus, content_transformer(removeWords),
stopwords("SMART"))


# Create a document-term matrix and remove sparse terms
testing_DTM = DocumentTermMatrix(testing_corpus)
testing_DTM = removeSparseTerms(testing_DTM, 0.96)
test_DTM = as.matrix(testing_DTM)


# Naive Bayes model


NBmodel <- naiveBayes(train_DTM, as.factor(train_labels), laplace = 1)
```

```
NBresults <- predict(NBmodel, test_DTM)


NBresults_table <- as.data.frame(table(NBresults, test_labels))

tough_authors <- subset(NBresults_table, NBresults_table$Freq < 5 &
NBresults_table$NBresults == NBresults_table$test_labels)


confusion_matrix = confusionMatrix(table(NBresults, train_labels))

confusion_matrix$overall

##    Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
## AccuracyPValue

##      0.2488000      0.2334694      0.2319556      0.2662348      0.0200000
## 0.0000000

## McnemarPValue

##           NaN


# Boosting

test_and_train <- c(testing_DTM, training_DTM)

labels <- c(train_labels, test_labels)


train_container <- create_container(test_and_train,
as.numeric(factor(labels)), trainSize = 1:2500, testSize = 2501:5000,
virgin=FALSE)

boosting <- train_model(train_container,"BOOSTING")

bresults <- classify_model(train_container, boosting)

analytics <- create_analytics(train_container, bresults, b = 1)

summary(analytics)


## ENSEMBLE SUMMARY

##        n-ENSEMBLE COVERAGE n-ENSEMBLE RECALL

## n >= 1                   1              0.51

## ALGORITHM PERFORMANCE
## LOGITBOOST_PRECISION    LOGITBOOST_RECALL    LOGITBOOST_FSCORE

##               0.5546               0.5064               0.5130
```

I built two models using the C50train directory in an attempt to predict the author identities of the articles in the C50test directory.

To create these models, I first had to create a training corpus and a test corpus. Then, I performed some preprocessing such as removing numbers and stop words.

The first model used Naive Bayes and Laplace smoothing to account for words in the test set that were not in the training set. This model had a ~25% accuracy in predicting on the out-of-sample set. Specifically, the model was not able to accurately predict any of the articles for Darren Schuettle, Edna Fernandes, Jane Macartney, Mark Benedeich, Mure Dickie, Sarah Davison, Scott Hillis, and William Kazer.

The second model used boosting and had ~55% precision in predicting the correct author, with an F-Score of 0.513.

Based on the accuracy of the two models, I prefer boosting. Although, the boosting model took significantly longer to run than did the Naive Bayes.

Practice with association rule mining

```
library(arules)

## Loading required package: Matrix
##
## Attaching package: 'arules'
##
## The following objects are masked from 'package:base':
##
##     %in%, write

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

groceryrules <- apriori(groceries, parameter = list(support=.005,
confidence=.01, maxlen=4))

##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##        0.01    0.1    1 none FALSE            TRUE   0.005      1      4
##  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 [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [2138 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
top_lift = subset(groceryrules, lift > 3.7)
inspect(top_lift)
```

```
##    lhs                        rhs                      support confidence
lift
## 1  {herbs}                 => {root vegetables}    0.007015760 0.43125000
3.956477
## 2  {root vegetables}       => {herbs}              0.007015760 0.06436567
3.956477
## 3  {ham}                   => {white bread}        0.005083884 0.19531250
4.639851
## 4  {white bread}           => {ham}                0.005083884 0.12077295
4.639851
## 5  {berries}               => {whipped/sour cream} 0.009049314 0.27217125
3.796886
## 6  {whipped/sour cream} => {berries}              0.009049314 0.12624113
3.796886
## 7  {other vegetables,
##     root vegetables}       => {onions}             0.005693950 0.12017167
3.875044
## 8  {butter,
##     other vegetables}      => {whipped/sour cream} 0.005795628 0.28934010
4.036397
## 9  {whipped/sour cream,
##     whole milk}            => {butter}             0.006710727 0.20820189
3.757185
## 10 {citrus fruit,
##     pip fruit}             => {tropical fruit}     0.005592272 0.40441176
3.854060
## 11 {citrus fruit,
##     tropical fruit}        => {pip fruit}          0.005592272 0.28061224
3.709437
## 12 {other vegetables,
##     pip fruit,
##     whole milk}            => {root vegetables}    0.005490595 0.40601504
3.724961
## 13 {citrus fruit,
##     other vegetables,
##     whole milk}            => {root vegetables}    0.005795628 0.44531250
```

```
4.085493
## 14 {root vegetables,
##     whole milk,
##     yogurt}             => {tropical fruit}     0.005693950 0.39160839
3.732043
## 15 {other vegetables,
##     tropical fruit,
##     whole milk}          => {root vegetables}    0.007015760 0.41071429
3.768074
```

```
top_support_confidence = subset(groceryrules, support > .01 & confidence >
0.5)
inspect(top_support_confidence)
```

```
##    lhs                     rhs                    support confidence
lift
## 1  {curd,
##     yogurt}             => {whole milk}       0.01006609  0.5823529
2.279125
## 2  {butter,
##     other vegetables}   => {whole milk}       0.01148958  0.5736041
2.244885
## 3  {domestic eggs,
##     other vegetables}   => {whole milk}       0.01230300  0.5525114
2.162336
## 4  {whipped/sour cream,
##     yogurt}             => {whole milk}       0.01087951  0.5245098
2.052747
## 5  {other vegetables,
##     whipped/sour cream} => {whole milk}       0.01464159  0.5070423
1.984385
## 6  {other vegetables,
##     pip fruit}          => {whole milk}       0.01352313  0.5175097
2.025351
## 7  {citrus fruit,
##     root vegetables}    => {other vegetables} 0.01037112  0.5862069
3.029608
## 8  {root vegetables,
##     tropical fruit}     => {other vegetables} 0.01230300  0.5845411
3.020999
## 9  {root vegetables,
##     tropical fruit}     => {whole milk}       0.01199797  0.5700483
2.230969
## 10 {tropical fruit,
##     yogurt}             => {whole milk}       0.01514997  0.5173611
2.024770
## 11 {root vegetables,
##     yogurt}             => {whole milk}       0.01453991  0.5629921
2.203354
## 12 {rolls/buns,
##     root vegetables}    => {other vegetables} 0.01220132  0.5020921
2.594890
## 13 {rolls/buns,
##     root vegetables}    => {whole milk}       0.01270971  0.5230126
2.046888
## 14 {other vegetables,
##     yogurt}             => {whole milk}       0.02226741  0.5128806
2.007235
```

First, I set relatively low thresholds for the association rules (support = 0.005 and confidence = 0.01), meaning that combinations must exist in at least 0.5% of all transactions and that out of all transactions with the items on the left, the item on the right must be in at least 1% of them.

Then, I was interested in seeing if there were any association rules within this group with a particularly high lift. This can inform product placement and other marketing strategies such as coupons. For instance, we can put profitable/high-margin products next to specific items where the combination of the items has a high lift. From this subset, we might consider putting ham and white bread near each other and berries and whipped/sour cream next to each other since both produce and meat usually have high markups.

Finally, I was interested in looking at the most commonly purchased combination of items, where the combination of items on the left is in more than 1% of all transactions, and the item on the right is in more than 50% of those transactions. The only items that showed up on the right were whole milk and other vegetables, which is not too surprising since those are commonly purchased items in general. We can see that in this subset, tropical fruit and yogurt had the highest support with a 0.52 confidence when paired with milk. While it may not make sense to place these products together, one possible way to use this information could be to look into which specific tropical fruits are purchased along with yogurt and milk to create a new smoothie line.