STA380 Homework 2 Barton, Jace

Jace Barton

August 19, 2015

First, I load the libraries I will need.

```
library(RCurl)

## Warning: package 'RCurl' was built under R version 3.0.3

## Loading required package: bitops

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.0.3

library(reshape)

## Warning: package 'reshape' was built under R version 3.0.3

library(plyr)

## Warning: package 'plyr' was built under R version 3.0.3

##
## Attaching package: 'plyr'
##
## The following objects are masked from 'package:reshape':
##
##     rename, round_any

library(tm)

## Warning: package 'tm' was built under R version 3.0.3

library(caret)

## Warning: package 'caret' was built under R version 3.0.3

## Loading required package: lattice

library(kknn)

## Warning: package 'kknn' was built under R version 3.0.3

##
## Attaching package: 'kknn'
##
## The following object is masked from 'package:caret':
##
##     contr.dummy
```

```r
library(e1071)

## Warning: package 'e1071' was built under R version 3.0.3

library(randomForest)

## Warning: package 'randomForest' was built under R version 3.0.3

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

library(arules)

## Warning: package 'arules' was built under R version 3.0.3

## Loading required package: Matrix

## Warning: package 'Matrix' was built under R version 3.0.3

##
## Attaching package: 'Matrix'
##
## The following object is masked from 'package:reshape':
##
##     expand
##
## The following objects are masked from 'package:base':
##
##     crossprod, tcrossprod
##
##
## Attaching package: 'arules'
##
## The following objects are masked from 'package:tm':
##
##     dissimilarity, inspect
##
## The following objects are masked from 'package:base':
##
##     %in%, write
```

I will run a random forest model in this homework. Thus, I will set the random seed so my results can be reproduced.

```r
set.seed(722)
```

Now I'm ready to begin.

Graph Creation

Flights at ABIA

First, I will load in the data for the analysis.

```
AirportURLString =
getURL("https://raw.githubusercontent.com/jacebarton/STA380/master/data/ABIA.
csv", ssl.verifypeer=0L, followlocation = 1L)
Airport = read.csv(text=AirportURLString)
summary(Airport)
```

```
##       Year          Month         DayofMonth      DayOfWeek
##  Min.   :2008   Min.   : 1.00   Min.   : 1.00   Min.   :1.000
##  1st Qu.:2008   1st Qu.: 3.00   1st Qu.: 8.00   1st Qu.:2.000
##  Median :2008   Median : 6.00   Median :16.00   Median :4.000
##  Mean   :2008   Mean   : 6.29   Mean   :15.73   Mean   :3.902
##  3rd Qu.:2008   3rd Qu.: 9.00   3rd Qu.:23.00   3rd Qu.:6.000
##  Max.   :2008   Max.   :12.00   Max.   :31.00   Max.   :7.000
##
##     DepTime       CRSDepTime      ArrTime       CRSArrTime
##  Min.   :   1   Min.   :  55   Min.   :   1   Min.   :   5
##  1st Qu.: 917   1st Qu.: 915   1st Qu.:1107   1st Qu.:1115
##  Median :1329   Median :1320   Median :1531   Median :1535
##  Mean   :1329   Mean   :1320   Mean   :1487   Mean   :1505
##  3rd Qu.:1728   3rd Qu.:1720   3rd Qu.:1903   3rd Qu.:1902
##  Max.   :2400   Max.   :2346   Max.   :2400   Max.   :2400
##  NA's   :1413                  NA's   :1567
##  UniqueCarrier     FlightNum        TailNum      ActualElapsedTime
##  WN     :34876   Min.   :   1          : 1104   Min.   : 22.0
##  AA     :19995   1st Qu.: 640   N678CA :  195   1st Qu.: 57.0
##  CO     : 9230   Median :1465   N511SW :  180   Median :125.0
##  YV     : 4994   Mean   :1917   N526SW :  176   Mean   :120.2
##  B6     : 4798   3rd Qu.:2653   N528SW :  172   3rd Qu.:164.0
##  XE     : 4618   Max.   :9741   N520SW :  168   Max.   :506.0
##  (Other):20749                  (Other):97265   NA's   :1601
##  CRSElapsedTime     AirTime          ArrDelay          DepDelay
##  Min.   : 17.0   Min.   :  3.00   Min.   :-129.000   Min.   :-42.000
##  1st Qu.: 58.0   1st Qu.: 38.00   1st Qu.:  -9.000   1st Qu.: -4.000
##  Median :130.0   Median :105.00   Median :  -2.000   Median :  0.000
##  Mean   :122.1   Mean   : 99.81   Mean   :   7.065   Mean   :  9.171
##  3rd Qu.:165.0   3rd Qu.:142.00   3rd Qu.:  10.000   3rd Qu.:  8.000
##  Max.   :320.0   Max.   :402.00   Max.   : 948.000   Max.   :875.000
##  NA's   :11      NA's   :1601     NA's   :1601       NA's   :1413
##      Origin            Dest           Distance        TaxiIn
##  AUS    :49623   AUS    :49637   Min.   :  66   Min.   :  0.000
##  DAL    : 5583   DAL    : 5573   1st Qu.: 190   1st Qu.:  4.000
##  DFW    : 5508   DFW    : 5506   Median : 775   Median :  5.000
##  IAH    : 3704   IAH    : 3691   Mean   : 705   Mean   :  6.413
##  PHX    : 2786   PHX    : 2783   3rd Qu.:1085   3rd Qu.:  7.000
```

```
##  DEN     : 2719   DEN     : 2673   Max.   :1770   Max.   :143.000
##  (Other):29337   (Other):29397                   NA's   :1567
##     TaxiOut         Cancelled       CancellationCode    Diverted
##  Min.   :  1.00   Min.   :0.00000    :97840          Min.   :0.000000
##  1st Qu.:  9.00   1st Qu.:0.00000   A:  719          1st Qu.:0.000000
##  Median : 12.00   Median :0.00000   B:  605          Median :0.000000
##  Mean   : 13.96   Mean   :0.01431   C:   96          Mean   :0.001824
##  3rd Qu.: 16.00   3rd Qu.:0.00000                    3rd Qu.:0.000000
##  Max.   :305.00   Max.   :1.00000                    Max.   :1.000000
##  NA's   :1419
##   CarrierDelay     WeatherDelay       NASDelay      SecurityDelay
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.00   Min.   :  0.00
##  1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.:  0.00
##  Median :  0.00   Median :  0.00   Median :  2.00   Median :  0.00
##  Mean   : 15.39   Mean   :  2.24   Mean   : 12.47   Mean   :  0.07
##  3rd Qu.: 16.00   3rd Qu.:  0.00   3rd Qu.: 16.00   3rd Qu.:  0.00
##  Max.   :875.00   Max.   :412.00   Max.   :367.00   Max.   :199.00
##  NA's   :79513    NA's   :79513    NA's   :79513    NA's   :79513
##  LateAircraftDelay
##  Min.   :  0.00
##  1st Qu.:  0.00
##  Median :  6.00
##  Mean   : 22.97
##  3rd Qu.: 30.00
##  Max.   :458.00
##  NA's   :79513
```

I immediately key in on delays as being the most interesting information in this dataset. For a given aircraft, are delays consistent for flights leaving Austin versus arriving in Austin?

To answer this, I first must split the dataset in two - one half for all of the flights leaving Austin, the other for all the flights arriving in Austin.

```
DepartureDelay <- data.frame(carrier=Airport$UniqueCarrier,
delay=Airport$DepDelay, leaving=Airport$Origin)
head(DepartureDelay)
```

```
##   carrier delay leaving
## 1      9E   345     MEM
## 2      AA    -5     AUS
## 3      YV     0     AUS
## 4      9E    -4     AUS
## 5      AA     1     AUS
## 6      NW    -9     AUS
```

This first step creates a data frame with all rows from the original data sets and columns for the airline, total delay for that flight, and the city from which the flight departed. I now want to filter this dataset to capture only Austin as the city of departure. I will also omit any rows where the departing city is unknown, as this means the flight was cancelled.

```
DepartureDelay = DepartureDelay[DepartureDelay$leaving == "AUS", ]
DepartureDelay = na.omit(DepartureDelay)
summary(DepartureDelay)
```

```
##     carrier          delay             leaving
##  WN     :17343   Min.   :-36.000   AUS    :48893
##  AA     : 9709   1st Qu.: -5.000   ABQ    :    0
##  CO     : 4554   Median : -1.000   ATL    :    0
##  YV     : 2456   Mean   :  7.425   BHM    :    0
##  B6     : 2367   3rd Qu.:  5.000   BNA    :    0
##  XE     : 2296   Max.   :875.000   BOS    :    0
##  (Other):10168                     (Other):    0
```

I can tell from the summary that this split the data almost exactly in half. I now need to aggregate delay information by carrier.

```
CarrierDepartureDelays = ddply(DepartureDelay, ~carrier, summarise,
mean=mean(delay), sd=sd(delay))
CarrierDepartureDelays
```

```
##    carrier      mean       sd
## 1       9E  3.656501 33.87182
## 2       AA  5.877536 28.25968
## 3       B6 10.451204 44.46459
## 4       CO  7.563900 32.73256
## 5       DL 12.099432 41.78871
## 6       EV 14.000000 40.72907
## 7       F9  1.599624 23.23526
## 8       MQ  7.820884 33.50115
## 9       NW  8.081967 48.06672
## 10      OH  9.926863 32.36067
## 11      OO  7.521761 33.37378
## 12      UA  5.833153 33.78858
## 13      US -0.778542 12.83104
## 14      WN  8.648158 24.97914
## 15      XE  5.597125 31.33525
## 16      YV  6.010586 35.25976
```

This completes my pre-processing for departing flights. I now need to do the same thing for arriving flights before final clean up.

```
ArrivalDelay <- data.frame(carrier=Airport$UniqueCarrier,
delay=Airport$ArrDelay, arriving=Airport$Dest)
head(DepartureDelay)
```

```
##   carrier delay leaving
## 2      AA    -5     AUS
## 3      YV     0     AUS
## 4      9E    -4     AUS
## 5      AA     1     AUS
## 6      NW    -9     AUS
## 7      CO    -9     AUS
```

```r
ArrivalDelay = ArrivalDelay[ArrivalDelay$arriving == "AUS", ]
ArrivalDelay = na.omit(ArrivalDelay)
summary(ArrivalDelay)
```

```
##     carrier          delay             arriving
##  WN     :17324   Min.   :-81.000   AUS    :48863
##  AA     : 9708   1st Qu.: -9.000   ABQ    :    0
##  CO     : 4555   Median : -1.000   ATL    :    0
##  YV     : 2467   Mean   :  8.091   BNA    :    0
##  B6     : 2365   3rd Qu.: 12.000   BOS    :    0
##  XE     : 2288   Max.   :518.000   BWI    :    0
##  (Other):10156                     (Other):    0
```

```r
CarrierArrivalDelays = ddply(ArrivalDelay, ~carrier, summarise,
mean=mean(delay), sd=sd(delay))
CarrierArrivalDelays
```

```
##    carrier      mean       sd
## 1       9E  3.518815 31.61852
## 2       AA  9.663473 34.46742
## 3       B6  9.610148 48.45161
## 4       CO  9.113063 35.08589
## 5       DL 12.979206 35.32323
## 6       EV 10.590571 38.63670
## 7       F9  5.172770 23.22650
## 8       MQ  6.428228 27.73061
## 9       NW 11.649123 49.13193
## 10      OH 15.274793 44.11506
## 11      OO  9.953854 36.11016
## 12      UA 12.237838 37.12183
## 13      US -2.640110 22.33975
## 14      WN  5.495324 29.89158
## 15      XE  6.173077 32.31526
## 16      YV 16.282529 47.60905
```

I now want to merge these two separate data frames. I also want to make the data more clear by using the airline name instead of the airline unique code.

```r
CarrierDelays = merge(CarrierArrivalDelays, CarrierDepartureDelays,
by="carrier")
CarrierDelays$CarrierNames = c("Pinnacle", "American", "JetBlue",
"Continental", "Delta", "AtlanticSE", "Frontier", "Envoy", "Northwest",
"Comair", "SkyWest", "United", "US", "Southwest", "ExpressJet", "Mesa")
CarrierDelays
```

```
##    carrier    mean.x     sd.x    mean.y     sd.y CarrierNames
## 1       9E  3.518815 31.61852  3.656501 33.87182     Pinnacle
## 2       AA  9.663473 34.46742  5.877536 28.25968     American
## 3       B6  9.610148 48.45161 10.451204 44.46459      JetBlue
## 4       CO  9.113063 35.08589  7.563900 32.73256  Continental
## 5       DL 12.979206 35.32323 12.099432 41.78871        Delta
```

```
## 6       EV 10.590571 38.63670 14.000000 40.72907   AtlanticSE
## 7       F9  5.172770 23.22650  1.599624 23.23526     Frontier
## 8       MQ  6.428228 27.73061  7.820884 33.50115        Envoy
## 9       NW 11.649123 49.13193  8.081967 48.06672    Northwest
## 10      OH 15.274793 44.11506  9.926863 32.36067       Comair
## 11      OO  9.953854 36.11016  7.521761 33.37378      SkyWest
## 12      UA 12.237838 37.12183  5.833153 33.78858       United
## 13      US -2.640110 22.33975 -0.778542 12.83104           US
## 14      WN  5.495324 29.89158  8.648158 24.97914    Southwest
## 15      XE  6.173077 32.31526  5.597125 31.33525   ExpressJet
## 16      YV 16.282529 47.60905  6.010586 35.25976         Mesa
```

Lastly, I will be interested in the total count of flights into and out of Austin for each airline. I will add this as a column. While I'm at it, I'll change the column names to be more meaningful.

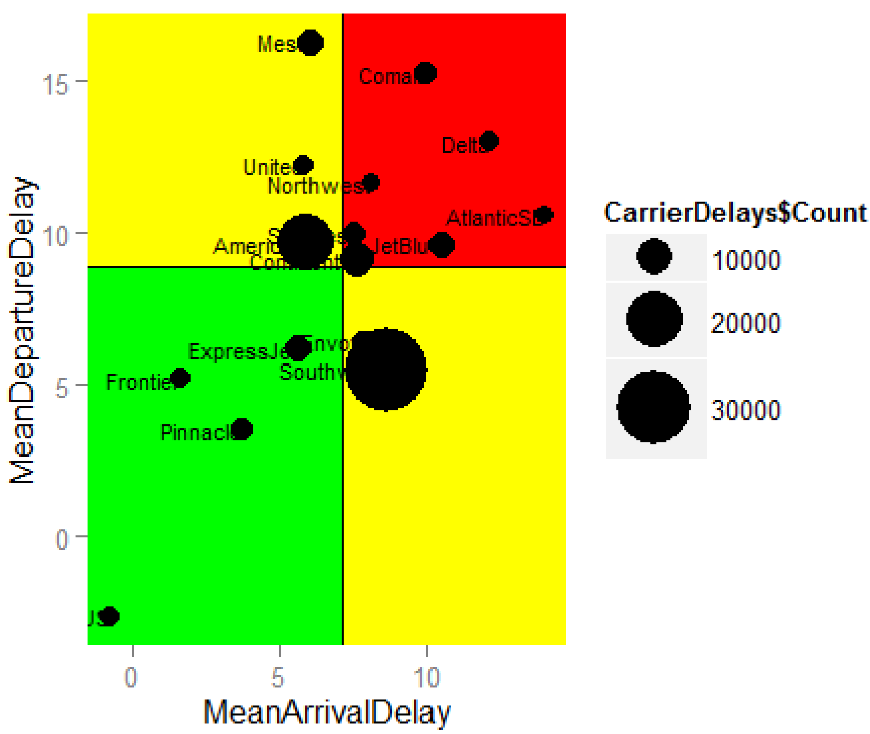
```
CarrierDelays$Count = summary(Airport$UniqueCarrier)
colnames(CarrierDelays) = c("CarrierCode", "MeanDepartureDelay",
"SDDepartureDelay", "MeanArrivalDelay", "SDArrivalDelay", "CarrierNames",
"Count")
CarrierDelays
```

```
##    CarrierCode MeanDepartureDelay SDDepartureDelay MeanArrivalDelay
## 1           9E           3.518815         31.61852         3.656501
## 2           AA           9.663473         34.46742         5.877536
## 3           B6           9.610148         48.45161        10.451204
## 4           CO           9.113063         35.08589         7.563900
## 5           DL          12.979206         35.32323        12.099432
## 6           EV          10.590571         38.63670        14.000000
## 7           F9           5.172770         23.22650         1.599624
## 8           MQ           6.428228         27.73061         7.820884
## 9           NW          11.649123         49.13193         8.081967
## 10          OH          15.274793         44.11506         9.926863
## 11          OO           9.953854         36.11016         7.521761
## 12          UA          12.237838         37.12183         5.833153
## 13          US          -2.640110         22.33975        -0.778542
## 14          WN           5.495324         29.89158         8.648158
## 15          XE           6.173077         32.31526         5.597125
## 16          YV          16.282529         47.60905         6.010586
##    SDArrivalDelay CarrierNames Count
## 1        33.87182     Pinnacle  2549
## 2        28.25968     American 19995
## 3        44.46459      JetBlue  4798
## 4        32.73256  Continental  9230
## 5        41.78871        Delta  2134
## 6        40.72907   AtlanticSE   825
## 7        23.23526     Frontier  2132
## 8        33.50115        Envoy  2663
## 9        48.06672    Northwest   121
## 10       32.36067       Comair  2986
```

```
## 11        33.37378       SkyWest  4015
## 12        33.78858        United  1866
## 13        12.83104            US  1458
## 14        24.97914     Southwest 34876
## 15        31.33525     ExpressJet  4618
## 16        35.25976          Mesa  4994
```

Now, what's all this been for? I want to get a picture of which airline is the best choice if I want to minimize my delays. I'd prefer my airline to be below average amongst all airlines in average delay on each leg of my trip - both departing and arriving. This can be seen in the following plot.

```
ggplot(CarrierDelays, aes(x=MeanArrivalDelay, y=MeanDepartureDelay,
label=CarrierNames)) + annotate("rect", xmin = -Inf, xmax =
mean(CarrierDelays$MeanArrivalDelay), ymin = -Inf, ymax =
mean(CarrierDelays$MeanDepartureDelay), fill= "green")  +
  annotate("rect", xmin = -Inf, xmax = mean(CarrierDelays$MeanArrivalDelay),
ymin = mean(CarrierDelays$MeanDepartureDelay), ymax = Inf , fill= "yellow") +
  annotate("rect", xmin = mean(CarrierDelays$MeanArrivalDelay), xmax = Inf,
ymin = -Inf, ymax = mean(CarrierDelays$MeanDepartureDelay), fill= "yellow") +
  annotate("rect", xmin = mean(CarrierDelays$MeanArrivalDelay), xmax = Inf,
ymin = mean(CarrierDelays$MeanDepartureDelay), ymax = Inf, fill= "red") +
  geom_point(aes(size=CarrierDelays$Count)) +
geom_vline(xintercept=mean(CarrierDelays$MeanArrivalDelay)) +
geom_hline(yintercept=mean(CarrierDelays$MeanDepartureDelay)) +
scale_size_continuous(range=c(3,15)) + geom_text(size=3, hjust=1)
```



Dots in the green square have smaller delays on average both arriving and departing, whereas dots in the red square have larger delays on average in both directions. The dot size is proportional to the number of flights the airline has into and out of Austin.

This graph can help a traveler determine which airline to take. For instance, if I want an airline with a lot of flights and good performance in getting me home on time, I'll choose American. On the other hand, if I want an airline with a lot of flights and I care most about getting to my non-Austin destination on time, I'll choose Southwest.

Text Analysis

Author Attribution

I am given approximately 50 New York Times articles each of 50 different authors as a training set, and another 50 atricles each of the same authors as a test set. Can I accurately predict which author a given article from the test set belongs to?

For simplicity, this analysis will ignore words from the test data set which are not in the training data set.

First, I pass in a function I will need to read the text data.

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

Now, I will bring in the test data, using the procedure from the example in class.

```
#Get all files
train_author_dirs = Sys.glob('../data/ReutersC50/C50train/*')
train_file_list = NULL
train_labels = NULL
#build a single corpus
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
train_all_docs = lapply(train_file_list, readerPlain)
names(train_all_docs) = sub('.txt', '', names(train_all_docs))

train_corpus = Corpus(VectorSource(train_all_docs))
names(train_corpus) = train_file_list

# Clean up tokens in corpus
train_corpus = tm_map(train_corpus, tolower) # make everything lowercase
```

```r
train_corpus = tm_map(train_corpus, removeNumbers) # remove numbers
train_corpus = tm_map(train_corpus, removePunctuation) # remove punctuation
train_corpus = tm_map(train_corpus, stripWhitespace) ## remove excess white-
space
train_corpus = tm_map(train_corpus, removeWords, stopwords("SMART"))

Train_Document_Term_Matrix = DocumentTermMatrix(train_corpus)
Train_Document_Term_Matrix # some basic summary statistics
```

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

```r
Train_Document_Term_Matrix = removeSparseTerms(Train_Document_Term_Matrix,
0.975)
tm:::inspect(Train_Document_Term_Matrix[1:10,1:5])
```

```
## A document-term matrix (10 documents, 5 terms)
##
## Non-/sparse entries: 3/47
## Sparsity           : 94%
## Maximal term length: 10
## Weighting          : term frequency (tf)
##
##     Terms
## Docs ability abroad access account accounting
##   1        0      0      1       0          0
##   2        0      0      0       0          0
##   3        0      0      2       0          0
##   4        0      0      0       0          0
##   5        0      0      0       0          0
##   6        0      0      0       0          0
##   7        0      0      0       0          0
##   8        0      0      0       0          0
##   9        0      0      0       0          0
##   10       0      0      4       0          0
```

```r
# Now a dense matrix
Train_Matrix = as.matrix(Train_Document_Term_Matrix)
```

I will repeat the above steps to build out the testing data.

```r
test_author_dirs = Sys.glob('../data/ReutersC50/C50test/*')
test_file_list = NULL
test_labels = NULL
#build a single corpus
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
test_all_docs = lapply(test_file_list, readerPlain)
names(test_all_docs) = sub('.txt', '', names(test_all_docs))

test_corpus = Corpus(VectorSource(test_all_docs))
names(test_corpus) = test_file_list

# Clean up tokens in corpus
test_corpus = tm_map(test_corpus, tolower) # make everything lowercase
test_corpus = tm_map(test_corpus, removeNumbers) # remove numbers
test_corpus = tm_map(test_corpus, removePunctuation) # remove punctuation
test_corpus = tm_map(test_corpus, stripWhitespace) ## remove excess white-
space
test_corpus = tm_map(test_corpus, removeWords, stopwords("SMART"))

Test_Document_Term_Matrix = DocumentTermMatrix(test_corpus, control =
list(dictionary=Terms(Train_Document_Term_Matrix)) )
Test_Document_Term_Matrix # some basic summary statistics

## A document-term matrix (2500 documents, 1389 terms)
##
## Non-/sparse entries: 246565/3225935
## Sparsity           : 93%
## Maximal term length: 18
## Weighting          : term frequency (tf)

tm:::inspect(Test_Document_Term_Matrix[1:10,1:5])

## A document-term matrix (10 documents, 5 terms)
##
## Non-/sparse entries: 9/41
## Sparsity           : 82%
## Maximal term length: 10
## Weighting          : term frequency (tf)
##
##     Terms
## Docs ability abroad access account accounting
##   1        0      0      0       1          3
##   2        0      0      0       0          0
##   3        0      0      3       0          0
##   4        1      1      0       0          0
##   5        0      0      0       0          0
##   6        0      0      0       0          0
##   7        0      0      0       0          0
##   8        1      0      4       0          0
```

```
##   9        1      0      4       0           0
##   10       0      0      0       0           0

# Now a dense matrix
Test_Matrix = as.matrix(Test_Document_Term_Matrix)
```

Now, I will build a Naive Bayes model to attempt to classify which test articles belong to which authors.

```
NaiveBayesModel = naiveBayes(Train_Matrix, as.factor(train_labels),
laplace=1)

NaiveBayesPredict = predict(object=NaiveBayesModel,newdata = Test_Matrix)
```

Now that the model is built, I can begin to look at results in different ways.

```
NaiveBayesResults = as.data.frame(table(NaiveBayesPredict,test_labels))
NaiveBayesResultsTable = table(NaiveBayesPredict, test_labels)
total_correct_vector = rep(0, 50)
predicted_per_author = rep(0,50)
for (i in 1:length(NaiveBayesResultsTable[1,])){
  total_correct_vector[i] = NaiveBayesResultsTable[i, i]
  predicted_per_author[i] = sum(NaiveBayesResultsTable[i,])
}

CorrectByAuthor = data.frame(row.names(NaiveBayesResultsTable),
total_correct_vector)
CorrectByAuthor

##    row.names.NaiveBayesResultsTable. total_correct_vector
## 1                        AaronPressman                   29
## 2                          AlanCrosby                   43
## 3                       AlexanderSmith                    0
## 4                      BenjaminKangLim                   16
## 5                        BernardHickey                    8
## 6                          BradDorfman                    0
## 7                     DarrenSchuettler                    0
## 8                          DavidLawder                   21
## 9                        EdnaFernandes                    0
## 10                         EricAuchard                    1
## 11                      FumikoFujisaki                    7
## 12                      GrahamEarnshaw                    1
## 13                    HeatherScoffield                    2
## 14                       JaneMacartney                    0
## 15                          JanLopatka                    6
## 16                        JimGilchrist                   39
## 17                            JoeOrtiz                    1
## 18                        JohnMastrini                    2
## 19                        JonathanBirt                    0
## 20                      JoWinterbottom                   18
## 21                         KarlPenhaul                    2
```

```
## 22                    KeithWeir                    1
## 23                KevinDrawbaugh                   0
## 24                 KevinMorrison                   1
## 25                 KirstinRidley                   0
## 26              KouroshKarimkhany                 40
## 27                     LydiaZajc                  47
## 28                LynneO'Donnell                  23
## 29               LynnleyBrowning                   4
## 30               MarcelMichelson                  12
## 31                  MarkBendeich                   0
## 32                    MartinWolk                   0
## 33                  MatthewBunce                   5
## 34                 MichaelConnor                   1
## 35                    MureDickie                   0
## 36                     NickLouth                   5
## 37               PatriciaCommins                   0
## 38                 PeterHumphrey                  32
## 39                    PierreTran                   2
## 40                    RobinSidel                  20
## 41                  RogerFillion                  35
## 42                   SamuelPerry                   1
## 43                  SarahDavison                   0
## 44                   ScottHillis                   0
## 45                   SimonCowell                   0
## 46                      TanEeLyn                   0
## 47                TheresePoletti                   6
## 48                    TimFarrand                  13
## 49                    ToddNissen                   0
## 50                  WilliamKazer                   0
```

```
PredictedForAuthor = data.frame(row.names(NaiveBayesResultsTable),
predicted_per_author)
SortedPredictedForAuthor = PredictedForAuthor[order(-
PredictedForAuthor$predicted_per_author),]
SortedPredictedForAuthor
```

```
##    row.names.NaiveBayesResultsTable. predicted_per_author
## 2                         AlanCrosby                  642
## 27                         LydiaZajc                  593
## 8                        DavidLawder                  354
## 26                 KouroshKarimkhany                  243
## 16                       JimGilchrist                 139
## 41                      RogerFillion                   80
## 4                     BenjaminKangLim                  75
## 38                     PeterHumphrey                   74
## 1                      AaronPressman                   59
## 48                        TimFarrand                   50
## 28                    LynneO'Donnell                   29
## 15                        JanLopatka                   28
## 20                    JoWinterbottom                   24
```

```
## 40                      RobinSidel                  21
## 30                 MarcelMichelson                  20
## 47                 TheresePoletti                   12
## 5                   BernardHickey                   11
## 11                 FumikoFujisaki                    8
## 36                      NickLouth                    8
## 33                   MatthewBunce                    5
## 13                HeatherScoffield                   4
## 29                 LynnleyBrowning                   4
## 21                    KarlPenhaul                    3
## 18                   JohnMastrini                    2
## 22                      KeithWeir                    2
## 34                  MichaelConnor                    2
## 39                     PierreTran                    2
## 10                    EricAuchard                    1
## 12                 GrahamEarnshaw                    1
## 17                       JoeOrtiz                    1
## 24                  KevinMorrison                    1
## 42                    SamuelPerry                    1
## 49                     ToddNissen                    1
## 3                  AlexanderSmith                    0
## 6                     BradDorfman                    0
## 7                DarrenSchuettler                    0
## 9                  EdnaFernandes                     0
## 14                  JaneMacartney                    0
## 19                   JonathanBirt                    0
## 23                KevinDrawbaugh                     0
## 25                 KirstinRidley                     0
## 31                  MarkBendeich                     0
## 32                     MartinWolk                    0
## 35                     MureDickie                    0
## 37                PatriciaCommins                    0
## 43                   SarahDavison                    0
## 44                    ScottHillis                    0
## 45                    SimonCowell                    0
## 46                       TanEeLyn                    0
## 50                   WilliamKazer                    0
```

```
OverallClassificationRate = sum(total_correct_vector)/2500
OverallClassificationRate
```

```
## [1] 0.1776
```

```
PrecisionRateByAuthor = data.frame(row.names(NaiveBayesResultsTable),
total_correct_vector/predicted_per_author)
PrecisionRateByAuthor
```

```
##    row.names.NaiveBayesResultsTable.
## 1                       AaronPressman
## 2                          AlanCrosby
## 3                      AlexanderSmith
```

```
## 4                  BenjaminKangLim
## 5                    BernardHickey
## 6                      BradDorfman
## 7                 DarrenSchuettler
## 8                      DavidLawder
## 9                   EdnaFernandes
## 10                    EricAuchard
## 11                 FumikoFujisaki
## 12                 GrahamEarnshaw
## 13               HeatherScoffield
## 14                  JaneMacartney
## 15                     JanLopatka
## 16                   JimGilchrist
## 17                       JoeOrtiz
## 18                   JohnMastrini
## 19                   JonathanBirt
## 20                 JoWinterbottom
## 21                    KarlPenhaul
## 22                      KeithWeir
## 23                 KevinDrawbaugh
## 24                  KevinMorrison
## 25                  KirstinRidley
## 26              KouroshKarimkhany
## 27                      LydiaZajc
## 28                 LynneO'Donnell
## 29                LynnleyBrowning
## 30                MarcelMichelson
## 31                   MarkBendeich
## 32                     MartinWolk
## 33                   MatthewBunce
## 34                  MichaelConnor
## 35                     MureDickie
## 36                      NickLouth
## 37                PatriciaCommins
## 38                  PeterHumphrey
## 39                     PierreTran
## 40                     RobinSidel
## 41                   RogerFillion
## 42                    SamuelPerry
## 43                   SarahDavison
## 44                    ScottHillis
## 45                    SimonCowell
## 46                       TanEeLyn
## 47                 TheresePoletti
## 48                     TimFarrand
## 49                     ToddNissen
## 50                   WilliamKazer
##    total_correct_vector.predicted_per_author
## 1                                 0.49152542
## 2                                 0.06697819
```

```
## 3                                              NaN
## 4                                       0.21333333
## 5                                       0.72727273
## 6                                              NaN
## 7                                              NaN
## 8                                       0.05932203
## 9                                              NaN
## 10                                      1.00000000
## 11                                      0.87500000
## 12                                      1.00000000
## 13                                      0.50000000
## 14                                             NaN
## 15                                      0.21428571
## 16                                      0.28057554
## 17                                      1.00000000
## 18                                      1.00000000
## 19                                             NaN
## 20                                      0.75000000
## 21                                      0.66666667
## 22                                      0.50000000
## 23                                             NaN
## 24                                      1.00000000
## 25                                             NaN
## 26                                      0.16460905
## 27                                      0.07925801
## 28                                      0.79310345
## 29                                      1.00000000
## 30                                      0.60000000
## 31                                             NaN
## 32                                             NaN
## 33                                      1.00000000
## 34                                      0.50000000
## 35                                             NaN
## 36                                      0.62500000
## 37                                             NaN
## 38                                      0.43243243
## 39                                      1.00000000
## 40                                      0.95238095
## 41                                      0.43750000
## 42                                      1.00000000
## 43                                             NaN
## 44                                             NaN
## 45                                             NaN
## 46                                             NaN
## 47                                      0.50000000
## 48                                      0.26000000
## 49                                      0.00000000
## 50                                             NaN
```

We achive a classification rate of 18.52%, which isn't superb. Moreover, there are 15 authors who we never predict to have written an article while there are 7 authors we predict to have written over 100 articles. In short, the results from Naive Bayes are inconsistent at best.

I will try a random forest model to see if I get better results.

```
TrainDataFrame = as.data.frame(Train_Matrix)
TestDataFrame = as.data.frame(Test_Matrix)

set.seed(722)
AuthorRandomForest = randomForest(x=TrainDataFrame,
y=as.factor(train_labels), ntree=50, mtry=30)

PredictedAuthor = predict(AuthorRandomForest, newdata = TestDataFrame)
```

I set the number of trees arbitrarily to be 50. The recomended number of variables to consider for categorical random forest problems is the square root of the number of predictor variables. In this case, that is the 1189 words remaining after the tokenization process. I round down from approximately 34 to arrive at 30 words. I then fit my random forest model to the test data set.

```
RandomForestResults = as.data.frame(table(PredictedAuthor,test_labels))
RandomForestResultsTable = table(PredictedAuthor, test_labels)
RF_total_correct_vector = rep(0, 50)
RF_predicted_per_author = rep(0,50)
for (i in 1:length(RandomForestResultsTable[1,])){
  RF_total_correct_vector[i] = RandomForestResultsTable[i, i]
  RF_predicted_per_author[i] = sum(RandomForestResultsTable[i,])
}

RFCorrectByAuthor =
data.frame(row.names(RandomForestResultsTable),RF_total_correct_vector)
RFCorrectByAuthor

##    row.names.RandomForestResultsTable. RF_total_correct_vector
## 1                         AaronPressman                      42
## 2                            AlanCrosby                      30
## 3                        AlexanderSmith                      19
## 4                       BenjaminKangLim                      16
## 5                         BernardHickey                      31
## 6                           BradDorfman                      29
## 7                     DarrenSchuettler                      14
## 8                           DavidLawder                       9
## 9                         EdnaFernandes                      19
## 10                          EricAuchard                      19
## 11                       FumikoFujisaki                      50
## 12                       GrahamEarnshaw                      43
## 13                     HeatherScoffield                      19
## 14                         JaneMacartney                       8
```

```
## 15                          JanLopatka                        32
## 16                        JimGilchrist                        50
## 17                            JoeOrtiz                        19
## 18                        JohnMastrini                        21
## 19                        JonathanBirt                        31
## 20                      JoWinterbottom                        37
## 21                         KarlPenhaul                        45
## 22                           KeithWeir                        34
## 23                      KevinDrawbaugh                        23
## 24                       KevinMorrison                        23
## 25                       KirstinRidley                        26
## 26                   KouroshKarimkhany                        34
## 27                           LydiaZajc                        32
## 28                      LynneO'Donnell                        40
## 29                     LynnleyBrowning                        49
## 30                     MarcelMichelson                        44
## 31                        MarkBendeich                        41
## 32                          MartinWolk                        22
## 33                        MatthewBunce                        46
## 34                       MichaelConnor                        27
## 35                          MureDickie                        16
## 36                           NickLouth                        40
## 37                     PatriciaCommins                        28
## 38                       PeterHumphrey                        29
## 39                          PierreTran                        21
## 40                          RobinSidel                        39
## 41                        RogerFillion                        39
## 42                         SamuelPerry                        22
## 43                        SarahDavison                        24
## 44                         ScottHillis                        18
## 45                         SimonCowell                        34
## 46                            TanEeLyn                        29
## 47                      TheresePoletti                        14
## 48                          TimFarrand                        23
## 49                          ToddNissen                        29
## 50                        WilliamKazer                        12
```

```
RFPredictedForAuthor = data.frame(row.names(RandomForestResultsTable),
RF_predicted_per_author)
RFSortedPredictedForAuthor = RFPredictedForAuthor[order(-
RFPredictedForAuthor$RF_predicted_per_author),]
RFSortedPredictedForAuthor
```

```
##    row.names.RandomForestResultsTable. RF_predicted_per_author
## 46                            TanEeLyn                        82
## 30                     MarcelMichelson                        79
## 19                        JonathanBirt                        74
## 21                         KarlPenhaul                        70
## 49                          ToddNissen                        69
## 26                   KouroshKarimkhany                        68
```

```
## 4                         BenjaminKangLim                         65
## 15                           JanLopatka                           65
## 12                        GrahamEarnshaw                          63
## 16                          JimGilchrist                          63
## 38                         PeterHumphrey                          61
## 31                          MarkBendeich                          60
## 1                          AaronPressman                          59
## 13                       HeatherScoffield                         59
## 17                             JoeOrtiz                           57
## 34                         MichaelConnor                          57
## 36                            NickLouth                           57
## 11                        FumikoFujisaki                          56
## 7                        DarrenSchuettler                         55
## 37                        PatriciaCommins                         55
## 6                           BradDorfman                           54
## 29                        LynnleyBrowning                         54
## 33                          MatthewBunce                          53
## 40                            RobinSidel                          53
## 45                           SimonCowell                          52
## 5                          BernardHickey                          49
## 41                          RogerFillion                          49
## 44                           ScottHillis                          49
## 9                          EdnaFernandes                          48
## 20                        JoWinterbottom                          48
## 23                        KevinDrawbaugh                          48
## 22                            KeithWeir                           47
## 42                           SamuelPerry                          44
## 18                          JohnMastrini                          43
## 28                        LynneO'Donnell                          42
## 2                             AlanCrosby                          41
## 10                           EricAuchard                          40
## 43                          SarahDavison                          40
## 48                            TimFarrand                          39
## 24                         KevinMorrison                          37
## 35                            MureDickie                          37
## 27                            LydiaZajc                           33
## 3                         AlexanderSmith                          32
## 25                         KirstinRidley                          32
## 50                          WilliamKazer                          32
## 47                        TheresePoletti                          29
## 32                            MartinWolk                          28
## 14                         JaneMacartney                          27
## 39                            PierreTran                          27
## 8                            DavidLawder                          19
```

```
RFOverallClassificationRate = sum(RF_total_correct_vector)/2500
RFOverallClassificationRate
```

```
## [1] 0.5764
```

```
RFPrecisionRateByAuthor = data.frame(row.names(RandomForestResultsTable),
RF_total_correct_vector/RF_predicted_per_author)
RFPrecisionRateByAuthor
```

```
##    row.names.RandomForestResultsTable.
## 1                          AaronPressman
## 2                            AlanCrosby
## 3                         AlexanderSmith
## 4                        BenjaminKangLim
## 5                          BernardHickey
## 6                            BradDorfman
## 7                       DarrenSchuettler
## 8                            DavidLawder
## 9                          EdnaFernandes
## 10                           EricAuchard
## 11                        FumikoFujisaki
## 12                        GrahamEarnshaw
## 13                      HeatherScoffield
## 14                         JaneMacartney
## 15                            JanLopatka
## 16                          JimGilchrist
## 17                              JoeOrtiz
## 18                          JohnMastrini
## 19                          JonathanBirt
## 20                        JoWinterbottom
## 21                           KarlPenhaul
## 22                             KeithWeir
## 23                        KevinDrawbaugh
## 24                         KevinMorrison
## 25                         KirstinRidley
## 26                     KouroshKarimkhany
## 27                             LydiaZajc
## 28                        LynneO'Donnell
## 29                       LynnleyBrowning
## 30                       MarcelMichelson
## 31                          MarkBendeich
## 32                            MartinWolk
## 33                          MatthewBunce
## 34                         MichaelConnor
## 35                            MureDickie
## 36                             NickLouth
## 37                       PatriciaCommins
## 38                         PeterHumphrey
## 39                            PierreTran
## 40                            RobinSidel
## 41                          RogerFillion
## 42                           SamuelPerry
## 43                          SarahDavison
## 44                           ScottHillis
## 45                           SimonCowell
```

```
## 46                                 TanEeLyn
## 47                           TheresePoletti
## 48                               TimFarrand
## 49                               ToddNissen
## 50                             WilliamKazer
##    RF_total_correct_vector.RF_predicted_per_author
## 1                                        0.7118644
## 2                                        0.7317073
## 3                                        0.5937500
## 4                                        0.2461538
## 5                                        0.6326531
## 6                                        0.5370370
## 7                                        0.2545455
## 8                                        0.4736842
## 9                                        0.3958333
## 10                                       0.4750000
## 11                                       0.8928571
## 12                                       0.6825397
## 13                                       0.3220339
## 14                                       0.2962963
## 15                                       0.4923077
## 16                                       0.7936508
## 17                                       0.3333333
## 18                                       0.4883721
## 19                                       0.4189189
## 20                                       0.7708333
## 21                                       0.6428571
## 22                                       0.7234043
## 23                                       0.4791667
## 24                                       0.6216216
## 25                                       0.8125000
## 26                                       0.5000000
## 27                                       0.9696970
## 28                                       0.9523810
## 29                                       0.9074074
## 30                                       0.5569620
## 31                                       0.6833333
## 32                                       0.7857143
## 33                                       0.8679245
## 34                                       0.4736842
## 35                                       0.4324324
## 36                                       0.7017544
## 37                                       0.5090909
## 38                                       0.4754098
## 39                                       0.7777778
## 40                                       0.7358491
## 41                                       0.7959184
## 42                                       0.5000000
## 43                                       0.6000000
## 44                                       0.3673469
```

```
## 45                                              0.6538462
## 46                                              0.3536585
## 47                                              0.4827586
## 48                                              0.5897436
## 49                                              0.4202899
## 50                                              0.3750000
```

I'm immediately much happier with my results. The overall classifcation rate jumps to 58%. Furthermore, there is a much better distribution of predicted number of articles for each author. This statistic ranges from 19 to 82 instead of 0 to 554 as in the Naive Bayes case.

Another nice thing about the Random Forest model is I can see which of the tokens were most important in fitting the model.

```
RFImportance = as.data.frame(importance(AuthorRandomForest))
RFImportanceDataFrame = data.frame(row.names(RFImportance), RFImportance)
SortRFImportance = RFImportanceDataFrame[order(-
RFImportanceDataFrame$MeanDecreaseGini),]
SortRFImportance[1:10,]
```

```
##            row.names.RFImportance. MeanDecreaseGini
## czech                        czech         18.81620
## toronto                    toronto         13.90261
## kong                          kong         13.04951
## hong                          hong         12.91905
## cargo                        cargo         12.87677
## french                      french         11.79951
## chinas                      chinas         11.51090
## australian              australian         11.28318
## china                        china         10.92082
## chinese                    chinese         10.84070
```

Interestingly, many international words appear on the list. This leads me to believe that the authors we were most succesfully able to classify write mostly about international items for the paper.

Association Rules

Practice with Association Rule Mining

First, I read in the data directly as transaction data and view a summary.

```
Groceries = read.transactions("../data/groceries.txt", format = "basket",
sep=",")
summary(Groceries)
```

```
## transactions as itemMatrix in sparse format with
##  9835 rows (elements/itemsets/transactions) and
##  169 columns (items) and a density of 0.02609146
```

```
##
## most frequent items:
##       whole milk other vegetables       rolls/buns             soda
##             2513             1903             1809             1715
##           yogurt          (Other)
##             1372            34055
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55
##   16   17   18   19   20   21   22   23   24   26   27   28   29   32
##   46   29   14   14    9   11    4    6    1    1    1    1    3    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   4.409   6.000  32.000
##
## includes extended item information - examples:
##             labels
## 1 abrasive cleaner
## 2 artif. sweetener
## 3   baby cosmetics
```

Whole milk is the most popular item, followed by a generic vegetable category and a generic bread category.

Next, I will create association rules for these transactions using arbitrary cutoffs for support, confidence, and number of items allowed in a rule. I will explore the cutoffs in more detail next.

```
GroceriesRules <- apriori(Groceries, parameter=list(support=.005,
confidence=.5, maxlen=4))
```

```
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.5    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 ... [120 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
arules:::inspect(GroceriesRules)
```

```
##     lhs                            rhs                     support
confidence     lift
## 1   {baking powder}             => {whole milk}       0.009252669
0.5229885 2.046793
## 2   {oil,
##      other vegetables}          => {whole milk}       0.005083884
0.5102041 1.996760
## 3   {onions,
##      root vegetables}           => {other vegetables} 0.005693950
0.6021505 3.112008
## 4   {onions,
##      whole milk}                => {other vegetables} 0.006609049
0.5462185 2.822942
## 5   {hygiene articles,
##      other vegetables}          => {whole milk}       0.005185562
0.5425532 2.123363
## 6   {other vegetables,
##      sugar}                     => {whole milk}       0.006304016
0.5849057 2.289115
## 7   {long life bakery product,
##      other vegetables}          => {whole milk}       0.005693950
0.5333333 2.087279
## 8   {cream cheese ,
##      yogurt}                    => {whole milk}       0.006609049
0.5327869 2.085141
## 9   {chicken,
##      root vegetables}           => {other vegetables} 0.005693950
0.5233645 2.704829
## 10  {chicken,
##      root vegetables}           => {whole milk}       0.005998983
0.5514019 2.157993
## 11  {chicken,
##      rolls/buns}                => {whole milk}       0.005287239
0.5473684 2.142208
## 12  {coffee,
##      yogurt}                    => {whole milk}       0.005083884
0.5208333 2.038359
## 13  {frozen vegetables,
##      root vegetables}           => {other vegetables} 0.006100661
0.5263158 2.720082
## 14  {frozen vegetables,
##      root vegetables}           => {whole milk}       0.006202339
0.5350877 2.094146
## 15  {frozen vegetables,
##      rolls/buns}                => {whole milk}       0.005083884
```

```
0.5000000 1.956825
## 16  {frozen vegetables,
##      other vegetables}         => {whole milk}       0.009659380
0.5428571 2.124552
## 17  {beef,
##      yogurt}                   => {whole milk}       0.006100661
0.5217391 2.041904
## 18  {beef,
##      rolls/buns}               => {whole milk}       0.006812405
0.5000000 1.956825
## 19  {curd,
##      whipped/sour cream}       => {whole milk}       0.005897306
0.5631068 2.203802
## 20  {curd,
##      tropical fruit}           => {yogurt}           0.005287239
0.5148515 3.690645
## 21  {curd,
##      tropical fruit}           => {other vegetables} 0.005287239
0.5148515 2.660833
## 22  {curd,
##      tropical fruit}           => {whole milk}       0.006507372
0.6336634 2.479936
## 23  {curd,
##      root vegetables}          => {other vegetables} 0.005490595
0.5046729 2.608228
## 24  {curd,
##      root vegetables}          => {whole milk}       0.006202339
0.5700935 2.231146
## 25  {curd,
##      yogurt}                   => {whole milk}       0.010066090
0.5823529 2.279125
## 26  {curd,
##      rolls/buns}               => {whole milk}       0.005897306
0.5858586 2.292845
## 27  {curd,
##      other vegetables}         => {whole milk}       0.009862735
0.5739645 2.246296
## 28  {pork,
##      root vegetables}          => {other vegetables} 0.007015760
0.5149254 2.661214
## 29  {pork,
##      root vegetables}          => {whole milk}       0.006812405
0.5000000 1.956825
## 30  {pork,
##      rolls/buns}               => {whole milk}       0.006202339
0.5495495 2.150744
## 31  {frankfurter,
##      tropical fruit}           => {whole milk}       0.005185562
0.5483871 2.146195
## 32  {frankfurter,
```

```
##      root vegetables}          => {whole milk}      0.005083884
0.5000000 1.956825
## 33  {frankfurter,
##      yogurt}                   => {whole milk}      0.006202339
0.5545455 2.170296
## 34  {bottled beer,
##      yogurt}                   => {whole milk}      0.005185562
0.5604396 2.193364
## 35  {brown bread,
##      tropical fruit}           => {whole milk}      0.005693950
0.5333333 2.087279
## 36  {brown bread,
##      root vegetables}          => {whole milk}      0.005693950
0.5600000 2.191643
## 37  {brown bread,
##      other vegetables}         => {whole milk}      0.009354347
0.5000000 1.956825
## 38  {domestic eggs,
##      margarine}                => {whole milk}      0.005185562
0.6219512 2.434099
## 39  {margarine,
##      root vegetables}          => {other vegetables} 0.005897306
0.5321101 2.750028
## 40  {margarine,
##      rolls/buns}               => {whole milk}      0.007930859
0.5379310 2.105273
## 41  {butter,
##      domestic eggs}            => {whole milk}      0.005998983
0.6210526 2.430582
## 42  {butter,
##      whipped/sour cream}       => {other vegetables} 0.005795628
0.5700000 2.945849
## 43  {butter,
##      whipped/sour cream}       => {whole milk}      0.006710727
0.6600000 2.583008
## 44  {butter,
##      citrus fruit}             => {whole milk}      0.005083884
0.5555556 2.174249
## 45  {bottled water,
##      butter}                   => {whole milk}      0.005388917
0.6022727 2.357084
## 46  {butter,
##      tropical fruit}           => {other vegetables} 0.005490595
0.5510204 2.847759
## 47  {butter,
##      tropical fruit}           => {whole milk}      0.006202339
0.6224490 2.436047
## 48  {butter,
##      root vegetables}          => {other vegetables} 0.006609049
0.5118110 2.645119
```

```
## 49  {butter,
##      root vegetables}          => {whole milk}       0.008235892
0.6377953 2.496107
## 50  {butter,
##      yogurt}                   => {whole milk}       0.009354347
0.6388889 2.500387
## 51  {butter,
##      other vegetables}         => {whole milk}       0.011489578
0.5736041 2.244885
## 52  {newspapers,
##      root vegetables}          => {other vegetables} 0.005998983
0.5221239 2.698417
## 53  {newspapers,
##      root vegetables}          => {whole milk}       0.005795628
0.5044248 1.974142
## 54  {domestic eggs,
##      whipped/sour cream}       => {other vegetables} 0.005083884
0.5102041 2.636814
## 55  {domestic eggs,
##      whipped/sour cream}       => {whole milk}       0.005693950
0.5714286 2.236371
## 56  {domestic eggs,
##      pip fruit}                => {whole milk}       0.005388917
0.6235294 2.440275
## 57  {citrus fruit,
##      domestic eggs}            => {whole milk}       0.005693950
0.5490196 2.148670
## 58  {domestic eggs,
##      tropical fruit}           => {whole milk}       0.006914082
0.6071429 2.376144
## 59  {domestic eggs,
##      root vegetables}          => {other vegetables} 0.007320793
0.5106383 2.639058
## 60  {domestic eggs,
##      root vegetables}          => {whole milk}       0.008540925
0.5957447 2.331536
## 61  {domestic eggs,
##      yogurt}                   => {whole milk}       0.007727504
0.5390071 2.109485
## 62  {domestic eggs,
##      other vegetables}         => {whole milk}       0.012302999
0.5525114 2.162336
## 63  {fruit/vegetable juice,
##      root vegetables}          => {other vegetables} 0.006609049
0.5508475 2.846865
## 64  {fruit/vegetable juice,
##      root vegetables}          => {whole milk}       0.006507372
0.5423729 2.122657
## 65  {fruit/vegetable juice,
##      yogurt}                   => {whole milk}       0.009456024
```

```
0.5054348 1.978094
## 66  {pip fruit,
##      whipped/sour cream}       => {other vegetables} 0.005592272
0.6043956 3.123610
## 67  {pip fruit,
##      whipped/sour cream}       => {whole milk}       0.005998983
0.6483516 2.537421
## 68  {citrus fruit,
##      whipped/sour cream}       => {other vegetables} 0.005693950
0.5233645 2.704829
## 69  {citrus fruit,
##      whipped/sour cream}       => {whole milk}       0.006304016
0.5794393 2.267722
## 70  {sausage,
##      whipped/sour cream}       => {whole milk}       0.005083884
0.5617978 2.198679
## 71  {tropical fruit,
##      whipped/sour cream}       => {other vegetables} 0.007829181
0.5661765 2.926088
## 72  {tropical fruit,
##      whipped/sour cream}       => {whole milk}       0.007930859
0.5735294 2.244593
## 73  {root vegetables,
##      whipped/sour cream}       => {other vegetables} 0.008540925
0.5000000 2.584078
## 74  {root vegetables,
##      whipped/sour cream}       => {whole milk}       0.009456024
0.5535714 2.166484
## 75  {whipped/sour cream,
##      yogurt}                   => {whole milk}       0.010879512
0.5245098 2.052747
## 76  {rolls/buns,
##      whipped/sour cream}       => {whole milk}       0.007829181
0.5347222 2.092715
## 77  {other vegetables,
##      whipped/sour cream}       => {whole milk}       0.014641586
0.5070423 1.984385
## 78  {pip fruit,
##      sausage}                  => {whole milk}       0.005592272
0.5188679 2.030667
## 79  {pip fruit,
##      root vegetables}          => {other vegetables} 0.008134215
0.5228758 2.702304
## 80  {pip fruit,
##      root vegetables}          => {whole milk}       0.008947636
0.5751634 2.250988
## 81  {pip fruit,
##      yogurt}                   => {whole milk}       0.009557702
0.5310734 2.078435
## 82  {other vegetables,
```

```
##      pip fruit}                 => {whole milk}       0.013523132
0.5175097 2.025351
## 83  {pastry,
##      tropical fruit}            => {whole milk}       0.006710727
0.5076923 1.986930
## 84  {pastry,
##      root vegetables}           => {other vegetables} 0.005897306
0.5370370 2.775491
## 85  {pastry,
##      root vegetables}           => {whole milk}       0.005693950
0.5185185 2.029299
## 86  {pastry,
##      yogurt}                    => {whole milk}       0.009150991
0.5172414 2.024301
## 87  {citrus fruit,
##      root vegetables}           => {other vegetables} 0.010371124
0.5862069 3.029608
## 88  {citrus fruit,
##      root vegetables}           => {whole milk}       0.009150991
0.5172414 2.024301
## 89  {root vegetables,
##      shopping bags}             => {other vegetables} 0.006609049
0.5158730 2.666112
## 90  {sausage,
##      tropical fruit}            => {whole milk}       0.007219115
0.5182482 2.028241
## 91  {root vegetables,
##      sausage}                   => {whole milk}       0.007727504
0.5170068 2.023383
## 92  {root vegetables,
##      tropical fruit}            => {other vegetables} 0.012302999
0.5845411 3.020999
## 93  {root vegetables,
##      tropical fruit}            => {whole milk}       0.011997966
0.5700483 2.230969
## 94  {tropical fruit,
##      yogurt}                    => {whole milk}       0.015149975
0.5173611 2.024770
## 95  {root vegetables,
##      yogurt}                    => {other vegetables} 0.012913066
0.5000000 2.584078
## 96  {root vegetables,
##      yogurt}                    => {whole milk}       0.014539908
0.5629921 2.203354
## 97  {rolls/buns,
##      root vegetables}           => {other vegetables} 0.012201322
0.5020921 2.594890
## 98  {rolls/buns,
##      root vegetables}           => {whole milk}       0.012709710
0.5230126 2.046888
```

```
## 99  {other vegetables,
##      yogurt}                    => {whole milk}       0.022267412
0.5128806 2.007235
## 100 {fruit/vegetable juice,
##      other vegetables,
##      yogurt}                    => {whole milk}       0.005083884
0.6172840 2.415833
## 101 {fruit/vegetable juice,
##      whole milk,
##      yogurt}                    => {other vegetables} 0.005083884
0.5376344 2.778578
## 102 {other vegetables,
##      root vegetables,
##      whipped/sour cream}        => {whole milk}       0.005185562
0.6071429 2.376144
## 103 {root vegetables,
##      whipped/sour cream,
##      whole milk}                => {other vegetables} 0.005185562
0.5483871 2.834150
## 104 {other vegetables,
##      whipped/sour cream,
##      yogurt}                    => {whole milk}       0.005592272
0.5500000 2.152507
## 105 {whipped/sour cream,
##      whole milk,
##      yogurt}                    => {other vegetables} 0.005592272
0.5140187 2.656529
## 106 {other vegetables,
##      pip fruit,
##      root vegetables}           => {whole milk}       0.005490595
0.6750000 2.641713
## 107 {pip fruit,
##      root vegetables,
##      whole milk}                => {other vegetables} 0.005490595
0.6136364 3.171368
## 108 {other vegetables,
##      pip fruit,
##      yogurt}                    => {whole milk}       0.005083884
0.6250000 2.446031
## 109 {pip fruit,
##      whole milk,
##      yogurt}                    => {other vegetables} 0.005083884
0.5319149 2.749019
## 110 {citrus fruit,
##      other vegetables,
##      root vegetables}           => {whole milk}       0.005795628
0.5588235 2.187039
## 111 {citrus fruit,
##      root vegetables,
##      whole milk}                => {other vegetables} 0.005795628
```

```
0.6333333 3.273165
## 112 {root vegetables,
##      tropical fruit,
##      yogurt}                    => {whole milk}       0.005693950
0.7000000 2.739554
## 113 {other vegetables,
##      root vegetables,
##      tropical fruit}            => {whole milk}       0.007015760
0.5702479 2.231750
## 114 {root vegetables,
##      tropical fruit,
##      whole milk}                => {other vegetables} 0.007015760
0.5847458 3.022057
## 115 {other vegetables,
##      tropical fruit,
##      yogurt}                    => {whole milk}       0.007625826
0.6198347 2.425816
## 116 {tropical fruit,
##      whole milk,
##      yogurt}                    => {other vegetables} 0.007625826
0.5033557 2.601421
## 117 {other vegetables,
##      root vegetables,
##      yogurt}                    => {whole milk}       0.007829181
0.6062992 2.372842
## 118 {root vegetables,
##      whole milk,
##      yogurt}                    => {other vegetables} 0.007829181
0.5384615 2.782853
## 119 {other vegetables,
##      rolls/buns,
##      root vegetables}           => {whole milk}       0.006202339
0.5083333 1.989438
## 120 {other vegetables,
##      rolls/buns,
##      yogurt}                    => {whole milk}       0.005998983
0.5221239 2.043410
```

These cutoffs yield 120 rules. While this is a lot to parse, I notice that a lot of the rules are used to predict when someone will buy whole milk. Since whole milk is the most common item, I want to look at lift in an attempt to "normalize" against how frequent an item is.

```
arules:::inspect(subset(GroceriesRules, subset=lift > 3))
```

```
##   lhs                        rhs                    support confidence
lift
## 1 {onions,
##    root vegetables}    => {other vegetables} 0.005693950  0.6021505
3.112008
## 2 {curd,
```

```
##    tropical fruit}     => {yogurt}           0.005287239  0.5148515
3.690645
## 3 {pip fruit,
##    whipped/sour cream} => {other vegetables} 0.005592272  0.6043956
3.123610
## 4 {citrus fruit,
##    root vegetables}    => {other vegetables} 0.010371124  0.5862069
3.029608
## 5 {root vegetables,
##    tropical fruit}     => {other vegetables} 0.012302999  0.5845411
3.020999
## 6 {pip fruit,
##    root vegetables,
##    whole milk}         => {other vegetables} 0.005490595  0.6136364
3.171368
## 7 {citrus fruit,
##    root vegetables,
##    whole milk}         => {other vegetables} 0.005795628  0.6333333
3.273165
## 8 {root vegetables,
##    tropical fruit,
##    whole milk}         => {other vegetables} 0.007015760  0.5847458
3.022057
```

There are 8 rules with a lift of more than 3, indicating they have predictive power beyond just predicting a popular item will be in a basket. These rules don't predict purchasing whole milk, but they do predict purchasing other vegetables, which was the second most common item. Many of the rules make sense intuitively. For instance, people who buy citrus fruit and root vegetables are already shopping for produce, so buying other vegetbales isn't a stretch of the imagination.

Let's now look at rules which occur in more than 1.5% of all transactions.

```
arules:::inspect(subset(GroceriesRules, subset=support > 0.015))
```

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

There are only two rules which occur more than 1.5% of the time, and both are used to predict whole milk. From this, we can see that if a customer buys yogurt and produce, they are also very likely to buy whole milk.

Finally, I want to look at the cases where the predicted item occurs most frequently with the predictor items.

```
arules:::inspect(subset(GroceriesRules, subset=support > .003 & confidence >
0.65))
```

```
##   lhs                        rhs              support confidence     lift
## 1 {butter,
##    whipped/sour cream} => {whole milk} 0.006710727      0.660 2.583008
## 2 {other vegetables,
##    pip fruit,
##    root vegetables}    => {whole milk} 0.005490595      0.675 2.641713
## 3 {root vegetables,
##    tropical fruit,
##    yogurt}             => {whole milk} 0.005693950      0.700 2.739554
```

All of the predicted items in this case involve whole milk. Again, produce and other dairy products show up. These rules all have high lift as well.