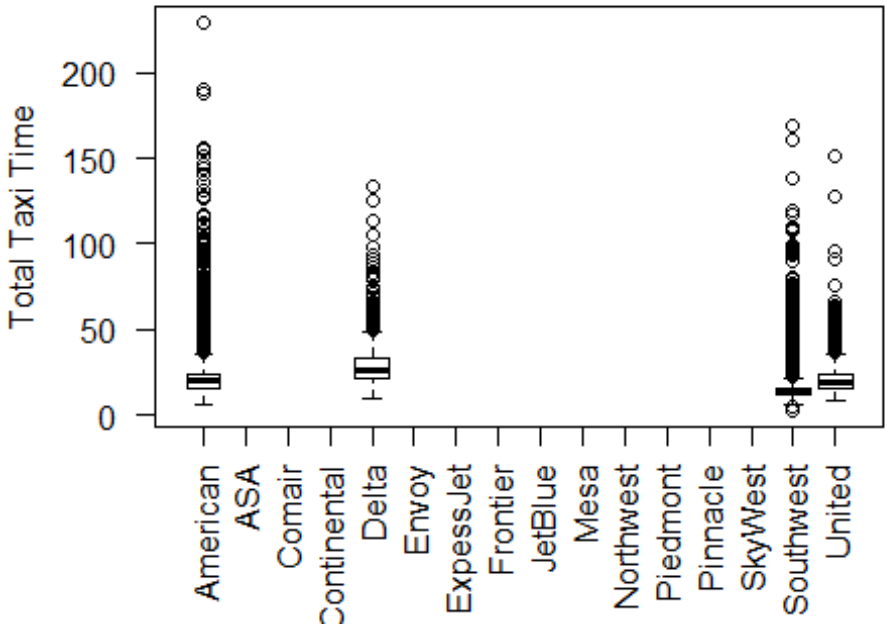
Exercise02

Bob Cook

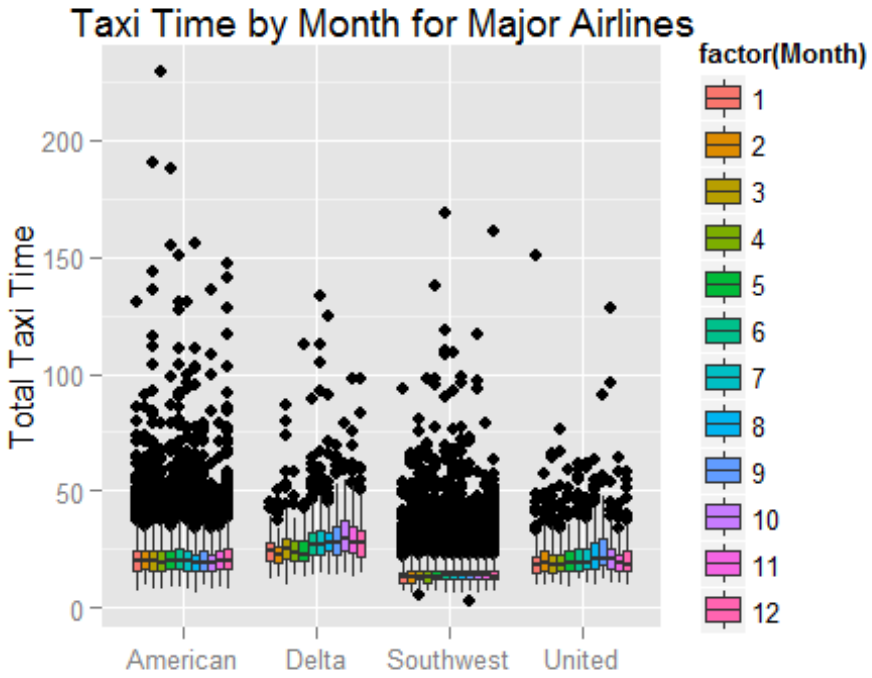
August 17, 2015

Flights at ABIA

```
carrier = c('Pinnacle', 'American', 'Mesa', 'Northwest', 'Continental',
'ExpessJet', 'JetBlue', 'Southwest', 'United', 'SkyWest', 'Comair', 'ASA',
'Piedmont', 'Envoy', 'Frontier', 'Delta')

code = c('9E', 'AA', 'YV', 'NW', 'CO', 'XE', 'B6', 'WN', 'UA', 'OO', 'OH',
'EV', 'US', 'MQ', 'F9', 'DL')

df = data.frame(carrier, code)

m1 = merge(abia, df, by.x = "UniqueCarrier", by.y = "code")
m1$TotalTaxiTime = m1$TaxiIn + m1$TaxiOut

m2 = subset(m1, m1$carrier == 'American' | m1$carrier == 'Southwest' |
m1$carrier == 'American' | m1$carrier == 'United' | m1$carrier == 'Delta' )


m2 = m1[m1$carrier == 'American' | m1$carrier == 'Southwest' | m1$carrier ==
'American' | m1$carrier == 'United' | m1$carrier == 'Delta', ]

plot(m2$TaxiOut+m2$TaxiIn~m2$carrier, par(las=2), xlab='', ylab='Total Taxi
Time')
```




```
m2$TotalTaxiTime = m2$TaxiIn + m2$TaxiOut

attach(m2)

## The following object is masked _by_ .GlobalEnv:
##
##     carrier

ggplot(m2, aes(carrier, TotalTaxiTime)) + geom_boxplot(aes(fill =
factor(Month))) + xlab('') + ylab('Total Taxi Time') + ggtitle('Taxi Time by
Month for Major Airlines')

## Warning: Removed 858 rows containing non-finite values (stat_boxplot).
```



Author Attribution

-Naive Bayes, Random Forest / PCA

First, we need to create two corpora, one for the training set and one for the test set. After the following pre-processing and (importantly) label creation, I have two Document-term matrices to run the classifications with.

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

## Warning: package 'e1071' was built under R version 3.2.2

## Loading required package: survival
## Loading required package: lattice
## Loading required package: splines
## Loading required package: parallel
## Loaded gbm 2.1.1
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:survival':
```

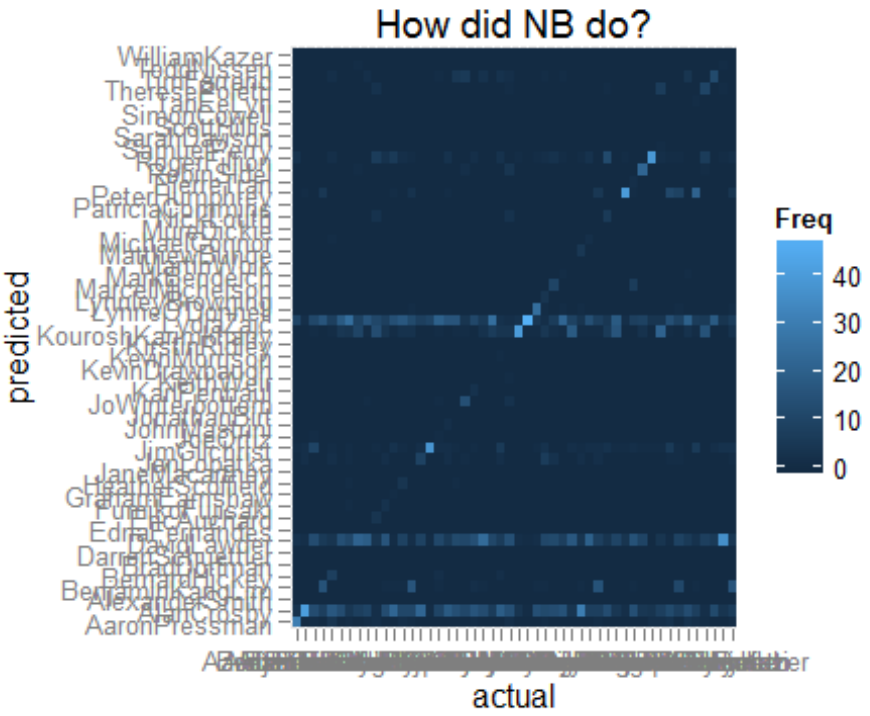
```
##
##     cluster
##
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

Next, before I can run classification, I have to convert these DTM's into data frames that can be recognized by the functions:

```
Xdf = as.data.frame(X)
Ydf = as.data.frame(Y)
```

Now, I run Naive Bayes, and produce a plot to show how the model did:

```
NB_1 = naiveBayes(x=Xdf, y=as.factor(labels_X), laplace=1)

predicted = predict(NB_1, Ydf)

tabled_NB = as.data.frame(table(predicted, labels_Y))

plotted_NB = ggplot(tabled_NB) + geom_tile(aes(x=labels_Y, y=predicted,
fill=Freq)) + ggtitle('How did NB do?') + xlab('actual')
plotted_NB
```



Naive Bayes did not perform too well. Now, let's check performance by author by sorting the sensitivity values in the confusion matrix:

```
NB_1_conf_matrix = confusionMatrix(table(predicted,labels_Y))

as.data.frame(NB_1_conf_matrix$byClass)[order(-
as.data.frame(NB_1_conf_matrix$byClass)$Sensitivity),1:2]
```

```
##                          Sensitivity Specificity
## Class: LydiaZajc                0.94   0.7930612
## Class: AlanCrosby               0.82   0.7930612
## Class: KouroshKarimkhany        0.80   0.9318367
## Class: PeterHumphrey            0.80   0.9746939
## Class: RogerFillion             0.78   0.9710204
## Class: JimGilchrist             0.76   0.9689796
## Class: AaronPressman            0.60   0.9873469
## Class: LynneO'Donnell           0.50   0.9975510
## Class: RobinSidel               0.44   0.9967347
## Class: DavidLawder              0.42   0.8244898
## Class: BenjaminKangLim          0.30   0.9714286
## Class: JoWinterbottom           0.28   0.9979592
## Class: TimFarrand               0.24   0.9877551
## Class: MarcelMichelson          0.20   0.9971429
## Class: TheresePoletti           0.20   0.9946939
## Class: JanLopatka               0.18   0.9881633
## Class: BernardHickey            0.16   0.9983673
## Class: NickLouth                0.12   0.9975510
## Class: MatthewBunce             0.10   1.0000000
## Class: EricAuchard              0.08   1.0000000
## Class: HeatherScoffield         0.08   0.9991837
## Class: LynnleyBrowning          0.08   1.0000000
## Class: FumikoFujisaki           0.06   0.9995918
## Class: KarlPenhaul              0.06   1.0000000
## Class: JohnMastrini             0.04   1.0000000
## Class: MichaelConnor            0.04   0.9995918
## Class: PierreTran               0.04   1.0000000
## Class: GrahamEarnshaw           0.02   1.0000000
## Class: JoeOrtiz                 0.02   0.9995918
## Class: KevinMorrison            0.02   1.0000000
## Class: MarkBendeich             0.02   0.9995918
## Class: SamuelPerry              0.02   1.0000000
## Class: TanEeLyn                 0.02   1.0000000
## Class: ToddNissen               0.02   0.9995918
## Class: AlexanderSmith           0.00   1.0000000
## Class: BradDorfman              0.00   1.0000000
## Class: DarrenSchuettler         0.00   1.0000000
## Class: EdnaFernandes            0.00   1.0000000
## Class: JaneMacartney            0.00   1.0000000
## Class: JonathanBirt             0.00   1.0000000
## Class: KeithWeir                0.00   0.9995918
## Class: KevinDrawbaugh           0.00   1.0000000
## Class: KirstinRidley            0.00   1.0000000
## Class: MartinWolk               0.00   1.0000000
```

```
## Class: MureDickie                0.00   1.0000000
## Class: PatriciaCommins           0.00   1.0000000
## Class: SarahDavison              0.00   1.0000000
## Class: ScottHillis               0.00   1.0000000
## Class: SimonCowell               0.00   1.0000000
## Class: WilliamKazer              0.00   1.0000000
```

So, for a few authors Naive Bayes performed okay, but overall it struggled to provide accuracy.

Random Forests: We'll run the random forests model and use the same method to check its prediction accuracy:

```
Y = as.matrix(Y)
X = as.matrix(X)

lhs <- data.frame(Y[,intersect(colnames(Y), colnames(X))])
rhs <- read.table(textConnection(""), col.names = colnames(X), colClasses =
"integer")
library(plyr)
cleaner_Y = rbind.fill(lhs, rhs)

cleaner_Ydf = as.data.frame(cleaner_Y)


RF_1 = randomForest(x=Xdf, y=as.factor(labels_X), mtry=4, ntree=150)

RF_1_predicted = predict(RF_1, data=cleaner_Y)


RF_1_conf_matrix = confusionMatrix(table(RF_1_predicted,labels_Y))

as.data.frame(RF_1_conf_matrix$byClass)[order(-
as.data.frame(RF_1_conf_matrix$byClass)$Sensitivity),1:2]
```

```
##                          Sensitivity Specificity
## Class: JimGilchrist             1.00   0.9930612
## Class: LynnleyBrowning          0.98   0.9971429
## Class: LynneO'Donnell           0.96   0.9942857
## Class: HeatherScoffield         0.94   0.9983673
## Class: LydiaZajc                0.94   0.9938776
## Class: RogerFillion             0.94   0.9959184
## Class: JoWinterbottom           0.90   0.9906122
## Class: FumikoFujisaki           0.88   0.9975510
## Class: PeterHumphrey            0.88   0.9861224
## Class: RobinSidel               0.88   0.9963265
## Class: DarrenSchuettler         0.86   0.9979592
## Class: DavidLawder              0.86   0.9934694
## Class: KouroshKarimkhany        0.86   0.9865306
## Class: MarcelMichelson          0.86   0.9971429
```

```
## Class: MatthewBunce              0.86   0.9967347
## Class: AaronPressman             0.84   0.9926531
## Class: JanLopatka                0.84   0.9914286
## Class: AlanCrosby                0.82   0.9959184
## Class: GrahamEarnshaw            0.82   0.9955102
## Class: BernardHickey             0.80   0.9938776
## Class: KarlPenhaul               0.78   0.9934694
## Class: MarkBendeich              0.78   0.9918367
## Class: MartinWolk                0.78   0.9975510
## Class: NickLouth                 0.78   0.9963265
## Class: ToddNissen                0.76   0.9946939
## Class: PierreTran                0.74   0.9979592
## Class: PatriciaCommins           0.72   0.9963265
## Class: TimFarrand                0.70   0.9918367
## Class: BenjaminKangLim           0.68   0.9800000
## Class: MichaelConnor             0.68   0.9951020
## Class: SimonCowell               0.68   0.9938776
## Class: JohnMastrini              0.64   0.9955102
## Class: AlexanderSmith            0.62   0.9946939
## Class: KevinDrawbaugh            0.62   0.9889796
## Class: KirstinRidley             0.62   0.9975510
## Class: TheresePoletti            0.60   0.9934694
## Class: EdnaFernandes             0.58   0.9897959
## Class: EricAuchard               0.58   0.9942857
## Class: JonathanBirt              0.56   0.9922449
## Class: KeithWeir                 0.56   0.9979592
## Class: KevinMorrison             0.56   0.9979592
## Class: JoeOrtiz                  0.52   0.9946939
## Class: SamuelPerry               0.50   0.9959184
## Class: BradDorfman               0.48   0.9967347
## Class: SarahDavison              0.46   0.9979592
## Class: MureDickie                0.38   0.9918367
## Class: JaneMacartney             0.36   0.9938776
## Class: TanEeLyn                  0.32   0.9914286
## Class: ScottHillis               0.28   0.9889796
## Class: WilliamKazer              0.24   0.9922449
```

Clearly, random forests performed much better than Naive Bayes, as expected.

Why the difference? In addition to advantages within the random forests algorithm, like the bootstrapping of the variables in addition to the documents, it seems to me that Naive Bayes' strong independence assumption between variables would be problematic in a NLP application. Certain words in a given document would likely be strongly correlated, which the Naive Bayes assumption skips over.

Practice with association rule mining

```
library(arules)

## Warning: package 'arules' was built under R version 3.2.2
```

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

```
groceries <- read.csv('C:/Users/Bob/Downloads/groceries.csv', header=FALSE)
groceries <- subset(groceries, V4 !='')
groceries_trans <- as(groceries, "transactions")
groceries_rules <- apriori(groceries_trans,
                      parameter=list(support=.001, confidence=.5, maxlen=4))
```

```
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.5    0.1    1 none FALSE            TRUE   0.001      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 ...[593 item(s), 7079 transaction(s)] done [0.00s].
## sorting and recoding items ... [406 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [167 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
summary(groceries_rules)
```

```
## set of 167 rules
##
## rule length distribution (lhs + rhs):sizes
##   2   3   4
##  23 130  14
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   3.000   3.000   2.946   3.000   4.000
##
## summary of quality measures:
```

```
##     support           confidence          lift
##  Min.   :0.001130   Min.   :0.5000   Min.   :  5.870
##  1st Qu.:0.001271   1st Qu.:0.5387   1st Qu.:  8.823
##  Median :0.001554   Median :0.6190   Median : 11.683
##  Mean   :0.002522   Mean   :0.6852   Mean   : 20.217
##  3rd Qu.:0.002401   3rd Qu.:0.8165   3rd Qu.: 17.660
##  Max.   :0.026275   Max.   :1.0000   Max.   :337.095
##
## mining info:
##             data ntransactions support confidence
##  groceries_trans          7079   0.001        0.5
```

```
# Look at the output
inspect(subset(groceries_rules, subset = support<.0015 & confidence > .8 &
lift>50))
```

```
##   lhs                      rhs                     support confidence
lift
## 1 {V2=bottled beer,
##    V3=liquor}           => {V4=red/blush wine} 0.001271366        0.9
127.42200
## 2 {V3=liquor,
##    V4=red/blush wine}   => {V2=bottled beer}   0.001271366        1.0
153.89130
## 3 {V2=bottled beer,
##    V4=red/blush wine}   => {V3=liquor}         0.001271366        1.0
337.09524
## 4 {V1=chicken,
##    V3=beef}             => {V2=pork}           0.001130103        1.0
54.45385
## 5 {V1=other vegetables,
##    V4=curd}             => {V3=butter}         0.001130103        1.0
51.29710
## 6 {V2=whole milk,
##    V4=curd}             => {V3=butter}         0.001412629        1.0
51.29710
## 7 {V1=whole milk,
##    V3=curd}             => {V2=butter}         0.001271366        1.0
56.63200
```

I chose a small support level because I was interested to see what the associations within item sets that were not as popular. I chose a confidence and lift value that were high enough to get the number of items down to a workable amount. These item sets make sense because of how similar the items are to others in its own set.

These associations indicate that it may be beneficial to have these items grouped together in a store. It is interesting that, in my experience, you may find the items in the lhs column, but the corresponding item(s) in the rhs column may not be in the same store.

Stores with this situation may be missing out on significant sales given how how great the assocations are between these items. For example, even though it may be expensive for a given store that sells beer and wine to get a permit to also sell liquor, it may be worth it because of how great the association is. Similarly, I have seen stores that sell chicken and beef but not necessarily pork. Given that, in this data set, every transaction that included chicken and beef also included pork, it would make a lot of sense to add pork to the customers' options.