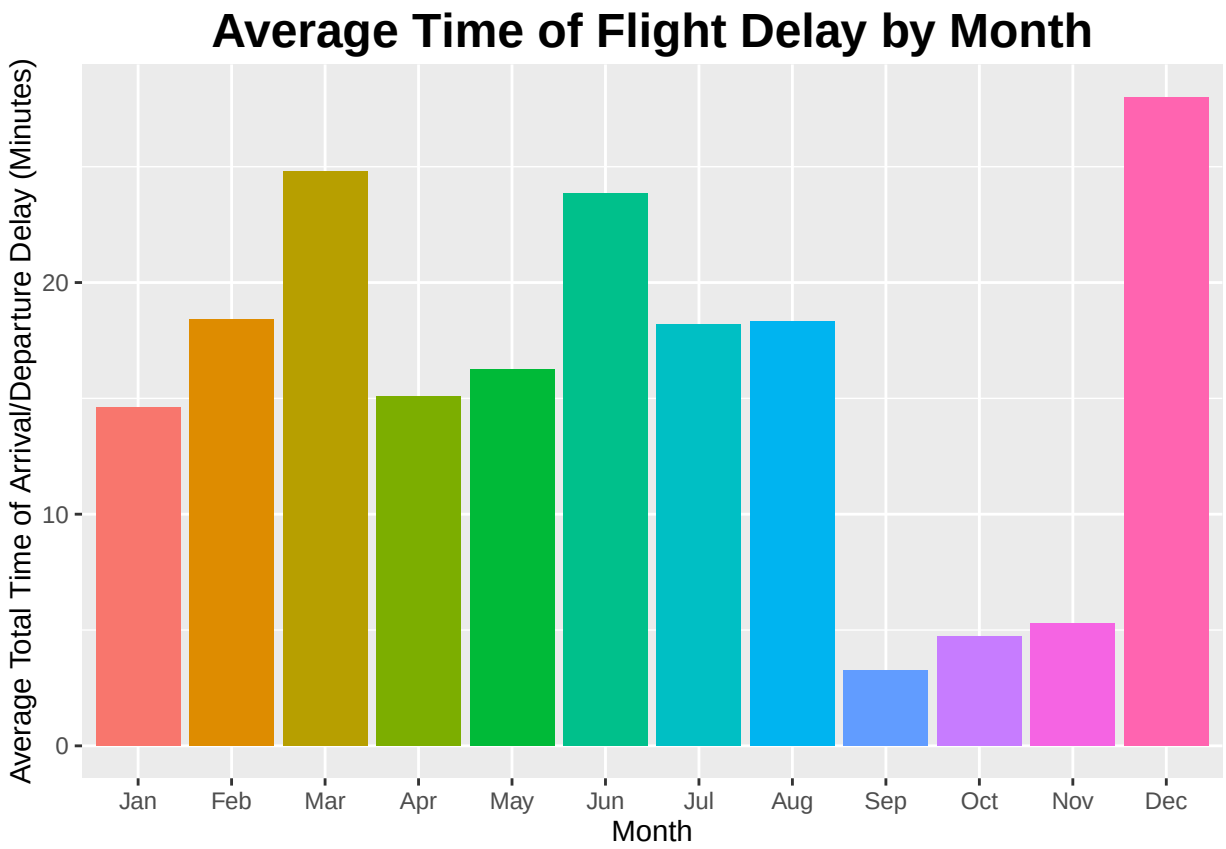
HW2_Graves_He_Yu_Zhu

Christina Graves, Mengnan He, Liz Zongyue Yu, Alice Yiqing Zhu

August 15, 2016

Question 1: Flights at ABIA - What is the best time of year to fly to minimize delays?



The above figure illustrates that the best time of year to travel in order to minimize arrival and departure flight delays would be during the fall months of September to November. The months in which travelers typically experience longer arrival and departure flight delays are December, March and June. Intuitively, this makes sense because December includes travel for the holidays, March includes travel for Spring Break, and June is the first month of summer vacation. These time periods typically see a influx of travellers, which could be a factor contributing to an increase in flight delays.

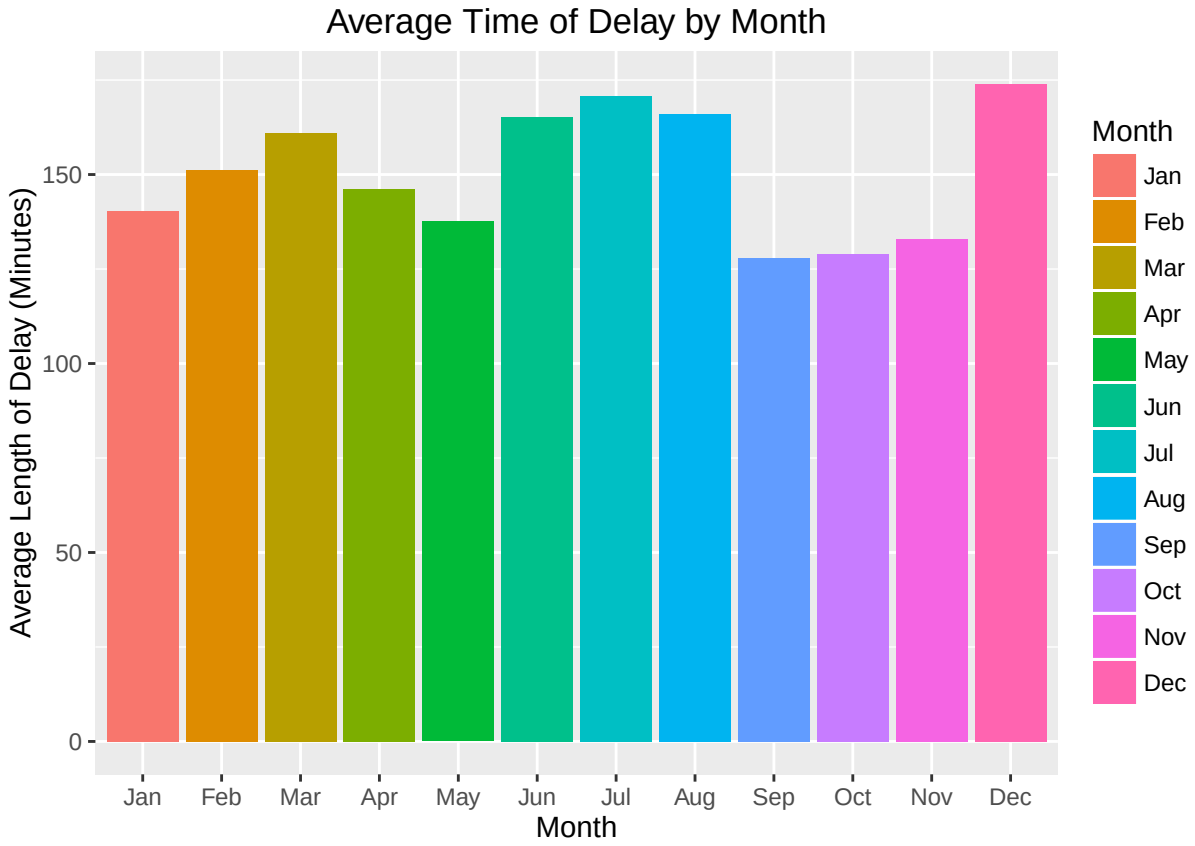
Note: These averages include only the departure and arrival delay times from the dataset due to several reasons. Because the other delay variables (Weather, Security, etc.) had a very large number of missing values and had very small variances in their delay times in comparison to the arrival and departure delay times, they were excluded from the delay computations in order to use as much of the dataset as possible. For example, the Security Delay variable had 79,513 NA values (80% of the total dataset) and a median of 0 minutes and mean delay time of .07 minutes (about 5 seconds), so including it would not significantly affect delay time per month.

When all of the delay variables are included with the exception of SecurityDelay, the months with the lowest average duration of delay time remained the same. The months with the highest delay times were December

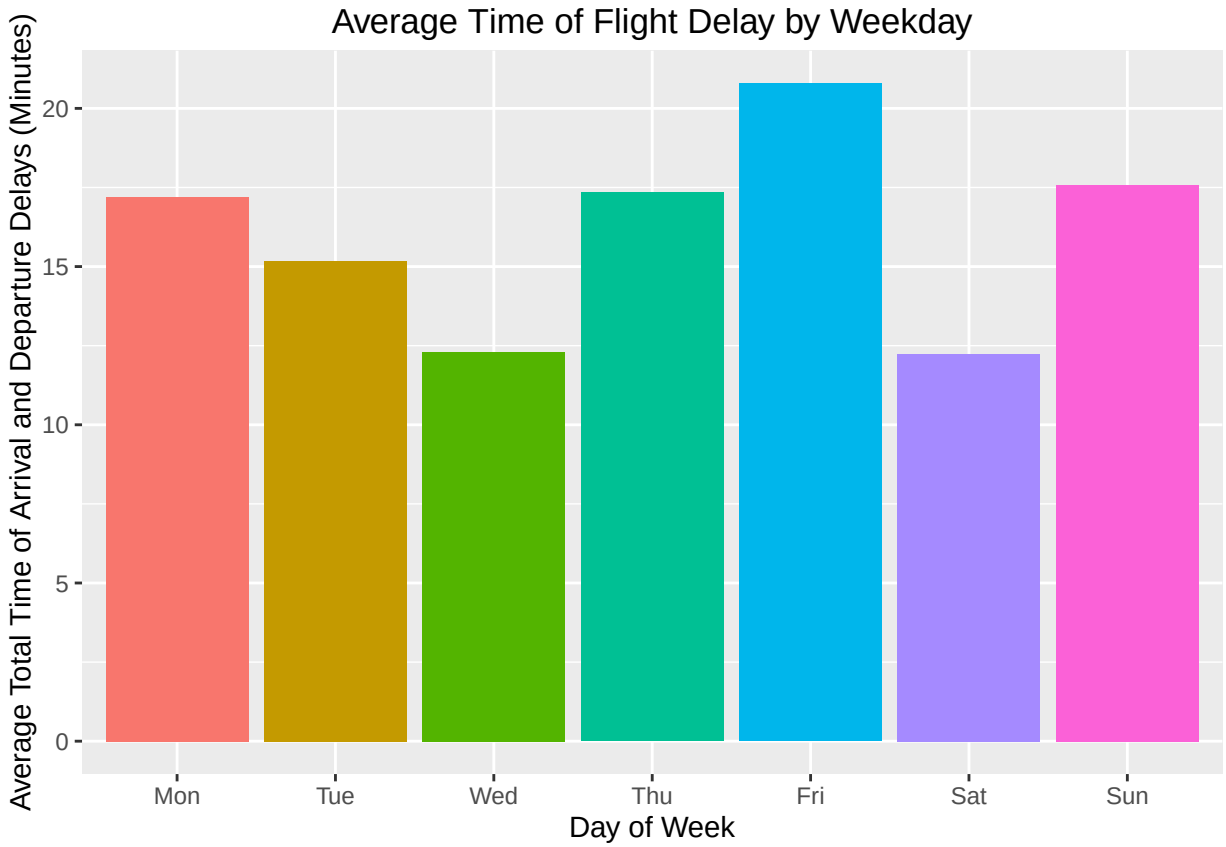
and the summer months of June, July and August. However, we believe this data is significantly skewed and misleading due to extreme outliers in some of the security variables as the month with the lowest average has an average delay time of about two hours, which does not seem reasonable. Additionally, by including all of these variables, we lose about 80% of the dataset due to missing values. Therefore, the original model above appeared to be the best figure in illustrating the best months for minimizing delay time.

This figure includes the following variables to calculate Average Length of Delay: Arrival Delay, Departure Delay, Carrier Delay, Weather Delay, NAS Delay and Late Aircraft Delay

```
## Warning: Removed 77912 rows containing non-finite values (stat_summary).
```



What is the best day of the week to travel to minimize delays?



In going a bit further, we also examined the best time of the week to travel. According to the figure above, Wednesday and Saturday have the lowest average delay times, while Friday is the worst day to travel to minimize delay time.

Question2: Text Classification

Model 1: Naive Bayes

For Reuter text, we need to make use of the tm package and the readerPlain function as below:

```
library(tm)
```

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

```
##
## Attaching package: 'NLP'
```

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

```
#use the readerPlain function in the Reuter example
readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }
```

We iterate over all 50 authors and read all texts from each one of them, gathering the corresponding author names as a list called "labels" while reading the texts.

```
## first get all author directories and then the corpus from each author
author_dirs = Sys.glob('ReutersC50/C50train/*')

file_list = NULL
labels = NULL

for(author in author_dirs) {
    author_name = substring(author, first=21)
    files_to_add = Sys.glob(paste0(author, '/*.txt'))
    file_list = append(file_list, files_to_add)
    labels = append(labels, rep(author_name, length(files_to_add)))
}
all_docs = lapply(file_list, readerPlain)
```

Then we build a corpus based on 2500 text files and pre-process them with tm_map() by removing numbers, punctuation, white spaces and stopwords.

```
# set reasonable names for each document
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))
my_corpus = Corpus(VectorSource(all_docs))
names(my_corpus) = file_list

# Preprocessing
my_corpus = tm_map(my_corpus, content_transformer(tolower)) # make everything lowercase
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers)) # remove numbers
my_corpus = tm_map(my_corpus, content_transformer(removePunctuation)) # remove punctuation
my_corpus = tm_map(my_corpus, content_transformer(stripWhitespace)) ## remove excess white-space
my_corpus = tm_map(my_corpus, content_transformer(removeWords), stopwords("SMART"))
```

Next we use the corpus above to form a document term matrix and remove some sparse terms.

```
DTM = DocumentTermMatrix(my_corpus)
DTM = removeSparseTerms(DTM, 0.99)
DTM
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 3076)>>
## Non-/sparse entries: 313587/7376413
## Sparsity           : 96%
## Maximal term length: 20
## Weighting          : term frequency (tf)
```

After getting the DTM, we start to build a train set based on it, grouping smoothed train records by authors.

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

# Naive Bayes: the training sets for the two authors
smooth_count = 1/nrow(X_train)
```

```
w_train = rowsum(X_train + smooth_count,labels)
w_train= w_train/sum(w_train)
w_train = log(w_train)
```

We redo every step for the test set before prediction:

```
#test
author_dirs = Sys.glob('ReutersC50/C50test/*')

file_list = NULL
labels_test = NULL
author_list = NULL

for(author in author_dirs) {
  author_name = substring(author, first=20)
  author_list = append(author_list,author_name)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  labels_test = append(labels_test, rep(author_name, length(files_to_add)))
}

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

test_corpus = Corpus(VectorSource(all_docs))
names(test_corpus) = file_list

# Preprocessing
test_corpus = tm_map(test_corpus, content_transformer(tolower)) # make everything lowercase
test_corpus = tm_map(test_corpus, content_transformer(removeNumbers)) # remove numbers
test_corpus = tm_map(test_corpus, content_transformer(removePunctuation)) # remove punctuation
test_corpus = tm_map(test_corpus, content_transformer(stripWhitespace)) ## remove excess white-space
test_corpus = tm_map(test_corpus, content_transformer(removeWords), stopwords("SMART"))

DTM_test = DocumentTermMatrix(test_corpus,list(dictionary=colnames(DTM)))
DTM_test
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 3076)>>
## Non-/sparse entries: 311938/7378062
## Sparsity           : 96%
## Maximal term length: 20
## Weighting          : term frequency (tf)
```

```
X_test = as.matrix(DTM_test)
```

As for prediction, we get maximum Naive Bayes log probabilities for each test row and use the author from the maximum probability as the predicted result.

```
predict = NULL
for (i in 1:nrow(X_test)) {
  max = -(Inf)
  author = NULL
  for (j in 1:nrow(w_train)) {
    result = sum(w_train[j,]*X_test[i,])
    if(result > max) {
      max = result
      author = rownames(w_train)[j]
    }
  }
  predict = append(predict, author)
}
```

Then we calculate the accuracy of Naive Bayes prediction by confusion matrix and adding all correct prediction for each author. We also calculate the correct rates.

```
predict_results = table(labels_test,predict)
correct = NULL
for (i in 1:nrow(predict_results)) {
  correct = append(correct, predict_results[i, i])
}

pred_table = data.frame(author_list, correct)
pred_table <- pred_table[order(-correct),]
pred_table$correct_rate <- pred_table$correct/50
```

From the result, we can see that 1/3 of all authors have really high correct rates (more than 80%), while Scott Hillis and Tan EeLyn only get below 20%.

```
pred_table
```

```
##          author_list correct correct_rate
## 29   LynnleyBrowning      49         0.98
## 11    FumikoFujisaki      48         0.96
## 16      JimGilchrist      48         0.96
## 36         NickLouth      46         0.92
## 38     PeterHumphrey      45         0.90
## 21       KarlPenhaul      44         0.88
## 33      MatthewBunce      44         0.88
## 22         KeithWeir      42         0.84
## 40        RobinSidel      42         0.84
## 3     AlexanderSmith      41         0.82
## 28    LynneO'Donnell      40         0.80
## 34     MichaelConnor      40         0.80
## 41      RogerFillion      40         0.80
## 1      AaronPressman      38         0.76
## 12    GrahamEarnshaw      38         0.76
## 6        BradDorfman      37         0.74
## 20      JonathanBirt      37         0.74
## 47    TheresePoletti      37         0.74
## 48        TimFarrand      36         0.72
```

```
## 30   MarcelMichelson      35         0.70
## 39        PierreTran      35         0.70
## 24     KevinMorrison      34         0.68
## 25     KirstinRidley      33         0.66
## 42       SamuelPerry      32         0.64
## 19      JohnMastrini      31         0.62
## 26 KouroshKarimkhany      31         0.62
## 27         LydiaZajc      31         0.62
## 43      SarahDavison      31         0.62
## 45       SimonCowell      31         0.62
## 18          JoeOrtiz      30         0.60
## 14        JanLopatka      28         0.56
## 10       EricAuchard      27         0.54
## 37   PatriciaCommins      27         0.54
## 23    KevinDrawbaugh      26         0.52
## 32        MartinWolk      25         0.50
## 2          AlanCrosby      23         0.46
## 5       BernardHickey      22         0.44
## 49         ToddNissen      22         0.44
## 17    JoWinterbottom      20         0.40
## 31      MarkBendeich      20         0.40
## 15     JaneMacartney      19         0.38
## 35        MureDickie      19         0.38
## 13  HeatherScoffield      17         0.34
## 7   DarrenSchuettler      14         0.28
## 4    BenjaminKangLim      12         0.24
## 50      WilliamKazer      12         0.24
## 8        DavidLawder      10         0.20
## 9       EdnaFernandes      10         0.20
## 44       ScottHillis       6         0.12
## 46          TanEeLyn       2         0.04
```

Overall, the accuracy for all prediction using Naive Bayes is 60.28%.

```
sum(pred_table$correct)/nrow(X_test)
```

```
## [1] 0.6028
```

Question3:

1. When runing the Apriori Algorithm for the first time, we look at the rules with support level >10% and at the same time loose the confidence restrictions to 0.0001.

By choosing this set of threshold, our goal is to examine the goods commonly shopped by the majority of customers.

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##       1e-04    0.1    1 none FALSE            TRUE     0.1      1      4
##  target   ext
##   rules FALSE
```

```
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [8 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
##   lhs    rhs                support   confidence lift
## 1 {}  => {bottled water}    0.1105236 0.1105236  1
## 2 {}  => {tropical fruit}   0.1049314 0.1049314  1
## 3 {}  => {root vegetables}  0.1089985 0.1089985  1
## 4 {}  => {soda}             0.1743772 0.1743772  1
## 5 {}  => {yogurt}           0.1395018 0.1395018  1
## 6 {}  => {rolls/buns}       0.1839349 0.1839349  1
## 7 {}  => {other vegetables} 0.1934926 0.1934926  1
## 8 {}  => {whole milk}       0.2555160 0.2555160  1
```

From the results above, a group of rules with only one item (i.e. an empty LHS) like {} => {canned beer} were created. These rules mean that no matter what other items are involved, the item in the RHS will appear with the probability given by the rule's confidence, which equals the support.

Grocery shoppers'consumption habits differ from each other largely, hence, there's no significantly dominant support number observed (all below 26%), but we can still detect some common trends. Diary products like whole milk and yogurt, vegetables include root vegetables and other vegetables, beverages include bottled water and soda, as well as rolls/buns ranked the top when compared to other goods. For example,25.5% of people buy whole milk no matter what other items are involved in their basket, which make sense since milk consuption has low elasticity and a lot of people needs it in their diet.

2. When we lower the support threshold to 0.05, other single items that have relatively high support numbers include: meat(beef,pork, sausage), diary products(butter, curd, domestic eggs,margarine,sour cream), staple food (rolls/buns,bread,pastry), beverages(beer,juice,coffee),newspapers,napkins and shopping bags.

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##       1e-04    0.1    1 none FALSE            TRUE    0.05      1      4
##  target   ext
##   rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
##
```

```
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [34 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
##    lhs                   rhs                     support    confidence
## 1  {}                 => {canned beer}           0.07768175 0.07768175
## 2  {}                 => {coffee}                0.05805796 0.05805796
## 3  {}                 => {beef}                  0.05246568 0.05246568
## 4  {}                 => {curd}                  0.05327911 0.05327911
## 5  {}                 => {napkins}               0.05236401 0.05236401
## 6  {}                 => {pork}                  0.05765125 0.05765125
## 7  {}                 => {frankfurter}           0.05897306 0.05897306
## 8  {}                 => {bottled beer}          0.08052872 0.08052872
## 9  {}                 => {brown bread}           0.06487036 0.06487036
## 10 {}                 => {margarine}             0.05856634 0.05856634
## 11 {}                 => {butter}                0.05541434 0.05541434
## 12 {}                 => {newspapers}            0.07981698 0.07981698
## 13 {}                 => {domestic eggs}         0.06344687 0.06344687
## 14 {}                 => {fruit/vegetable juice} 0.07229283 0.07229283
## 15 {}                 => {whipped/sour cream}    0.07168277 0.07168277
## 16 {}                 => {pip fruit}             0.07564820 0.07564820
## 17 {}                 => {pastry}                0.08896797 0.08896797
## 18 {}                 => {citrus fruit}          0.08276563 0.08276563
## 19 {}                 => {shopping bags}         0.09852567 0.09852567
## 20 {}                 => {sausage}               0.09395018 0.09395018
## 29 {yogurt}           => {whole milk}            0.05602440 0.40160350
## 30 {whole milk}       => {yogurt}                0.05602440 0.21925985
## 31 {rolls/buns}       => {whole milk}            0.05663447 0.30790492
## 32 {whole milk}       => {rolls/buns}            0.05663447 0.22164743
## 33 {other vegetables} => {whole milk}            0.07483477 0.38675775
## 34 {whole milk}       => {other vegetables}      0.07483477 0.29287704
##    lift
## 1  1.000000
## 2  1.000000
## 3  1.000000
## 4  1.000000
## 5  1.000000
## 6  1.000000
## 7  1.000000
## 8  1.000000
## 9  1.000000
## 10 1.000000
## 11 1.000000
## 12 1.000000
## 13 1.000000
## 14 1.000000
## 15 1.000000
## 16 1.000000
## 17 1.000000
## 18 1.000000
```

```
## 19 1.000000
## 20 1.000000
## 29 1.571735
## 30 1.571735
## 31 1.205032
## 32 1.205032
## 33 1.513634
## 34 1.513634
```

Most of the categories listed above (except shopping bags) are life necessities, with low elasticity, thus yield higher support fraction among all the other goods. Shopping bags having high support is understandable too. People tend to forget to bring their own shopping bag often.

By looking at other rules that appeared in pairs from above, we see some relationships between yogurt and whole milk, rolls/buns and whole milk, as well as vegetables and whole milk. Though milk is the hottest item among people's grocery shopping list, people who buy yogurt, rolls/buns or other vegetables are 1.4 times in average more likely to buy whole milk than other shoppers.

3. Next, we reset the thresholds, with a lower support level, a higher confidence level and lift level, in order to examine goods appeared less often in people's basket, but have stronger in-basket relationships.

In order to avoid rules that contains only one single item, we use the argument parameter: minlen=2.

After testing for different sets of thresholds, we choose to first present support level between 0.4% and 1%, confidence level above 60% and lift above 3. The results are shown below:

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.6    0.1    1 none FALSE            TRUE   0.004      2      4
##  target   ext
##   rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 39
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [126 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [40 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
##   lhs                     rhs                    support confidence     lift
## 1 {onions,
##    root vegetables}    => {other vegetables} 0.005693950  0.6021505 3.112008
## 2 {pip fruit,
##    whipped/sour cream} => {other vegetables} 0.005592272  0.6043956 3.123610
## 3 {pip fruit,
##    root vegetables,
```

```
##    whole milk}         => {other vegetables} 0.005490595  0.6136364 3.171368
## 4 {citrus fruit,
##    root vegetables,
##    tropical fruit}     => {other vegetables} 0.004473818  0.7857143 4.060694
## 5 {citrus fruit,
##    root vegetables,
##    whole milk}         => {other vegetables} 0.005795628  0.6333333 3.273165
## 6 {root vegetables,
##    tropical fruit,
##    yogurt}             => {other vegetables} 0.004982206  0.6125000 3.165495
```

Only 0.5% of customers buy sets of two to three items among root vegetables, oinions, pip fruit, tropical fruit,citrus fruit, whole milk, yogurt and sour creams. Among these people, 60% also buy other vegetables. Compared to others, they are 3.2 more likely to buy other vegetables. This reflects a group of people with healty diet style. They might also be vegetarians.

4. Next, we try to continue lowering support level in order to detect more rare but strongly related item sets. After trying for several times, we set the support level to (0.0006,0.001), with confidence level >0.6 and lift >15. This gives us a moderate number of rules to deal with.

A too strict confidence level(ie. close to 1) and a low support level yield too many rules. That's why we loose the confidence level to 0.6 and at the same time increase lift threshold.

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.6    0.1    1 none FALSE            TRUE   6e-04      2      4
##  target   ext
##   rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 5
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [164 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.02s].
## writing ... [8265 rule(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

```
##    lhs                           rhs                              support confidence     lift
## 1  {bottled beer,
##     popcorn}                  => {salty snack}             0.0007117438  0.7000000 18.50672
## 2  {hygiene articles,
##     Instant food products}    => {hamburger meat}          0.0006100661  0.8571429 25.77982
## 3  {butter,
##     Instant food products}    => {hamburger meat}          0.0008134215  0.6666667 20.05097
## 4  {domestic eggs,
##     Instant food products}    => {hamburger meat}          0.0006100661  0.6000000 18.04587
```

```
## 5  {Instant food products,
##     shopping bags}            => {hamburger meat}           0.0008134215  0.7272727 21.87378
## 6  {canned fish,
##     soft cheese}              => {cream cheese }            0.0006100661  0.8571429 21.61538
## 7  {dessert,
##     processed cheese}         => {white bread}              0.0007117438  0.6363636 15.11748
## 8  {ham,
##     specialty bar}            => {white bread}              0.0008134215  0.6666667 15.83736
## 9  {bottled beer,
##     liquor,
##     soda}                     => {red/blush wine}           0.0008134215  0.6666667 34.69136
## 10 {chocolate marshmallow,
##     waffles,
##     yogurt}                   => {margarine}                0.0006100661  1.0000000 17.07465
## 11 {chocolate marshmallow,
##     margarine,
##     yogurt}                   => {waffles}                  0.0006100661  0.8571429 22.30159
## 12 {chocolate marshmallow,
##     margarine,
##     other vegetables}         => {waffles}                  0.0006100661  0.8571429 22.30159
## 13 {chocolate marshmallow,
##     rolls/buns,
##     soda}                     => {waffles}                  0.0006100661  0.6000000 15.61111
## 14 {chocolate marshmallow,
##     rolls/buns,
##     whole milk}               => {waffles}                  0.0006100661  0.6666667 17.34568
## 15 {chocolate,
##     mustard,
##     whole milk}               => {oil}                      0.0006100661  0.7500000 26.72554
## 16 {long life bakery product,
##     mustard,
##     other vegetables}         => {chocolate}                0.0007117438  0.8750000 17.63448
## 17 {chocolate,
##     mustard,
##     other vegetables}         => {long life bakery product} 0.0007117438  1.0000000 26.72554
## 18 {canned fish,
##     domestic eggs,
##     other vegetables}         => {cream cheese }            0.0006100661  0.6666667 16.81197
## 19 {frankfurter,
##     ham,
##     processed cheese}         => {white bread}              0.0006100661  0.8571429 20.36232
## 20 {frankfurter,
##     processed cheese,
##     white bread}              => {ham}                      0.0006100661  0.6666667 25.61198
## 21 {frankfurter,
##     ham,
##     white bread}              => {processed cheese}         0.0006100661  0.6666667 40.22495
## 22 {fruit/vegetable juice,
##     ham,
##     processed cheese}         => {white bread}              0.0007117438  0.6363636 15.11748
## 23 {ham,
##     processed cheese,
##     shopping bags}            => {white bread}              0.0006100661  0.7500000 17.81703
## 24 {processed cheese,
```

```
##     shopping bags,
##     white bread}              => {ham}                       0.0006100661  0.6666667 25.61198
## 25 {ham,
##     processed cheese,
##     tropical fruit}           => {white bread}               0.0007117438  0.7777778 18.47692
## 26 {ham,
##     processed cheese,
##     yogurt}                   => {white bread}               0.0006100661  0.8571429 20.36232
## 27 {processed cheese,
##     white bread,
##     yogurt}                   => {ham}                       0.0006100661  0.6666667 25.61198
## 28 {ham,
##     other vegetables,
##     processed cheese}         => {white bread}               0.0008134215  0.7272727 17.27712
## 29 {other vegetables,
##     processed cheese,
##     white bread}              => {ham}                       0.0008134215  0.6153846 23.64183
## 30 {fruit/vegetable juice,
##     ham,
##     soda}                     => {processed cheese}          0.0006100661  0.6000000 36.20245
## 31 {processed cheese,
##     waffles,
##     whole milk}               => {white bread}               0.0006100661  0.6666667 15.83736
## 32 {domestic eggs,
##     processed cheese,
##     rolls/buns}               => {white bread}               0.0006100661  0.7500000 17.81703
## 33 {pastry,
##     processed cheese,
##     soda}                     => {white bread}               0.0006100661  0.6666667 15.83736
## 34 {ice cream,
##     pastry,
##     soda}                     => {salty snack}               0.0006100661  0.6000000 15.86290
## 35 {citrus fruit,
##     detergent,
##     other vegetables}         => {oil}                       0.0007117438  0.6363636 22.67622
## 36 {baking powder,
##     flour,
##     margarine}                => {sugar}                     0.0006100661  0.6666667 19.68969
## 37 {baking powder,
##     butter,
##     whipped/sour cream}       => {long life bakery product} 0.0006100661  0.6666667 17.81703
## 38 {chocolate,
##     flour,
##     whole milk}               => {sugar}                     0.0006100661  0.6666667 19.68969
## 39 {curd,
##     flour,
##     margarine}                => {sugar}                     0.0006100661  0.6666667 19.68969
## 40 {citrus fruit,
##     flour,
##     margarine}                => {sugar}                     0.0006100661  0.6000000 17.72072
## 41 {flour,
##     frankfurter,
##     other vegetables}         => {chicken}                   0.0006100661  0.7500000 17.47927
## 42 {other vegetables,
```

```
##     sliced cheese,
##     soft cheese}              => {ham}                  0.0006100661  0.6000000 23.05078
## 43 {curd,
##     root vegetables,
##     soft cheese}              => {cream cheese }        0.0006100661  0.6666667 16.81197
## 44 {butter milk,
##     ham,
##     whole milk}               => {dessert}              0.0007117438  0.6363636 17.14695
## 45 {butter milk,
##     long life bakery product,
##     pip fruit}                => {dessert}              0.0006100661  0.8571429 23.09589
## 46 {dessert,
##     long life bakery product,
##     pip fruit}                => {butter milk}          0.0006100661  0.7500000 26.82273
## 47 {butter milk,
##     long life bakery product,
##     yogurt}                   => {dessert}              0.0006100661  0.6666667 17.96347
## 48 {butter milk,
##     pip fruit,
##     white bread}              => {dessert}              0.0006100661  0.6666667 17.96347
## 49 {dessert,
##     pip fruit,
##     white bread}              => {butter milk}          0.0006100661  0.6000000 21.45818
## 50 {butter milk,
##     pip fruit,
##     whipped/sour cream}       => {dessert}              0.0006100661  0.6666667 17.96347
## 51 {dessert,
##     ham,
##     pip fruit}                => {curd}                 0.0006100661  0.8571429 16.08779
## 52 {curd,
##     ham,
##     pip fruit}                => {dessert}              0.0006100661  0.6666667 17.96347
## 53 {fruit/vegetable juice,
##     pip fruit,
##     sugar}                    => {cream cheese }        0.0006100661  0.6000000 15.13077
## 54 {canned beer,
##     soda,
##     waffles}                  => {chocolate}            0.0006100661  0.7500000 15.11527
## 55 {bottled water,
##     long life bakery product,
##     root vegetables}          => {butter}               0.0006100661  0.8571429 15.46789
## 56 {beef,
##     frankfurter,
##     whipped/sour cream}       => {cream cheese }        0.0006100661  0.6000000 15.13077
```

1) We see 0.07% of people have popcorn and bottled water in their basket. 70% of these people also buy salty snack and they are 18.5 times more likely to buy salty snack than others. Similar relationship was observed between ice cream, pastry, soda and salty snack. These people might be couch potatos that like to spend their leisure time eating popcorns and chips and watching TV.

2) Also, we see very strong relationships between instant food products,hamburger meat and hygiene articles/butter/eggs/shopping bags. These shoppers might be very busy and have no time cooking for themsleves. Even the occurance of shopping bags also makes sense. They might go grocery shopping right after work, thus didn't prepare their own shopping bag.

3) We can also observe a group of sweet lovers, they buy waffles, chocolate marshmallow, margarine and yogurt at the same time. Their lift is very high.

4) Sandwich/burger lovers are detected by a strong relationship between frankfurter, ham, processed cheese,tropical fruit, other vegetables, rolls/buns and white bread. This sounds reasonable since people make burgers or sandwiches using these ingredients.

5) 0.08% of people buy bottled beer, liquor and soda. 66% of them also buy red/blush wine at the same time. Alcohol lovers are 34.69 times more likely than others to have red/blush wine in their basket.

6) People who like bakery tend to have baking powder, flour, sugar, margarine,chocolate, butter, curd and long life bakery product in their basket. The lift level and confidence are very high, though support level was low.

7) Dairy and pip fruit lovers are also likely to be dessert lovers. Around 80% of people who buy butter milk, curd, pip fruit or whole milk also have dessert in their basket.

Conclusion

People's grocery shopping habits largely depend on their diet and life style, thus differ greatly from each other. By setting different sets of threshold, we can detect some common trends among people. However, there are still a lot of rules that we are unable to talk about due to the length of the report.