Clustering Grocery Items

Mitul Shah

8/16/2017

```r
## Loading the libraries
library(magrittr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
## Loading datasets
item_to_id <- read.csv("grocery/item_to_id.csv")
data <- read.csv("grocery/purchase_history.csv")

## Order the datasets
item_to_id <- item_to_id %>% arrange(Item_id)
data <- data %>% arrange(user_id)

## Loading splitstackshape
library(splitstackshape)
```

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, last
```

```r
## Creating dat
dat <- cSplit(data, "id", ",")

## Removing user id column
dat <- select(dat, -1)

## Creating grocery dataset
grocery <- data.frame(matrix(nrow = 39474, ncol = 48))
```

```r
## Renaming columns to items
colnames(grocery) <- item_to_id$Item_id

## Create x as sequence from 1 to 48
x <- seq(1,48,1)

## Enter 1 if the transaction had that item and 0 if that item was not in the transaction
for(i in 1:nrow(dat)) {
  for(j in 1:ncol(dat)) {
    if( (dat[[i,j]] %in% x) == TRUE) {
      y <- dat[[i, j]]
      grocery[[i, y]] = 1
    } else {
      grocery[[i, y]] = 0
    }
  }
}

## Enter 1 if the transaction had that item and 0 if that item was not in the transaction
#for(i in 1:nrow(dat)) {
#  for(j in 1:ncol(dat)) {
#    if(is.na(dat[[i,j]]) == FALSE)  {
#      y <- dat[[i, j]]
#      grocery[[i, y]] = 1
#    } else {
#      grocery[[i, y]] = 0
#    }
#  }
#}

grocery[grocery == 0] <- 1
grocery[is.na(grocery)] <- 0

## Renaming columns to Item name
colnames(grocery) <- item_to_id$Item_name
```

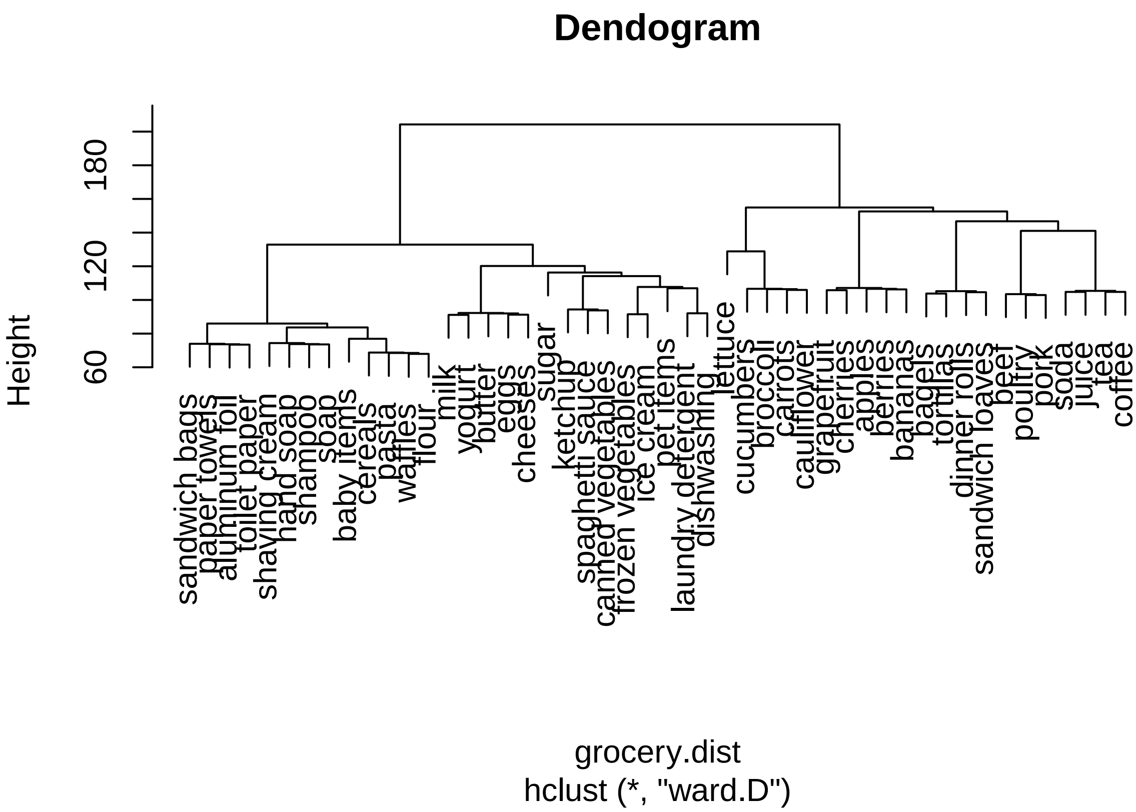
Forming Clusters of Grocery Items

```r
## Transpose grocery data
grocery_data_to_cluster <- as.data.frame(t(grocery))

## Distance matrix
grocery.dist <- dist(grocery_data_to_cluster, method = "euclidean", diag = FALSE, upper = FALSE, p = 2)

## Hierarchical Clustering using Ward's method
grocery.hclust <- hclust(grocery.dist, method = "ward.D")

## Visualize the dendogram
plot(grocery.hclust, labels = item_to_id$Item_name, main='Dendogram')
```



Looking at the dendogram, grouping items into 12 clusters seems to be a good number.

```
## Forming 12 clusters
groups.12 <- cutree(grocery.hclust, 12)

## Looking at the items in all 12 clusters
sapply(unique(groups.12), function(g)item_to_id$Item_name[groups.12 == g])
```

```
## [[1]]
## [1] sugar
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[2]]
## [1] lettuce
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[3]]
## [1] pet items         laundry detergent dishwashing
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[4]]
##  [1] baby items    waffles       sandwich bags cereals       shampoo
##  [6] aluminum foil shaving cream paper towels  hand soap     flour
## [11] pasta         toilet paper  soap
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[5]]
## [1] poultry beef    pork
```

```
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[6]]
## [1] butter  eggs    milk    cheeses yogurt
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[7]]
## [1] soda   tea    juice  coffee
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[8]]
## [1] carrots     cucumbers   broccoli    cauliflower
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[9]]
## [1] bagels          tortillas       dinner rolls    sandwich loaves
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[10]]
## [1] grapefruit cherries   apples     berries    bananas
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[11]]
## [1] frozen vegetables ice cream
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
##
## [[12]]
## [1] spaghetti sauce   canned vegetables ketchup
## 48 Levels: aluminum foil apples baby items bagels bananas beef ... yogurt
```

```r
## Comparing results with k-means forming 12 clusters
kmeans(grocery_data_to_cluster, 12, alg="Lloyd")[[1]]
```

```
##             sugar           lettuce         pet items        baby items
##                 5                 6                 7                 5
##           waffles           poultry     sandwich bags            butter
##                 5                 2                 2                 5
##              soda           carrots           cereals           shampoo
##                 2                 2                 2                 5
##            bagels              eggs     aluminum foil              milk
##                 5                 9                 2                 5
##              beef laundry detergent     shaving cream        grapefruit
##                10                 2                 5                 4
##           cheeses frozen vegetables               tea      paper towels
##                 5                 3                 1                 2
##          cherries   spaghetti sauce       dishwashing  canned vegetables
##                 5                11                12                 8
##         hand soap             flour             pasta            apples
##                 5                 5                 2                 5
##      toilet paper         tortillas              soap         ice cream
##                 2                 5                 5                 2
##      dinner rolls             juice   sandwich loaves           berries
##                 2                 2                 5                 2
##           ketchup         cucumbers            coffee          broccoli
```

```
##                 5                 2                 2                 6
##       cauliflower           bananas              pork            yogurt
##                 2                 5                 2                 5
```

The clusters formed by hierarchical clustering (Ward's method) seems to be better than the results shown by k-means clustering by looking at the names of the items in clusters. The results of k-means groups has 1 cluster with many items which doesn't look good to me. However, we might obtain better clusters by using k-mediods (using PAM) or using other linkage methods like single or complete linkage in hierarchical clustering. Looking at the above results, I would group the items by the results given by hierarchical clustering.

Finding customers who bought most items in her lifetime

```
## Merge user id with grocery
data_with_users <- as.data.frame(cbind(data$user_id, grocery))

## Rename 1st column to user id
colnames(data_with_users)[1] <- "user_id"

## Data giving number of items bought by each customer in each transaction
data_to_find_customers_buying_most_items <- data_with_users %>% mutate(total_items_in_each_transaction =

## Grouping by customer
number_of_items_by_customer <- data_to_find_customers_buying_most_items %>% group_by(user_id) %>% summa

## Finding maxiumum items bought by any customer
max(number_of_items_by_customer$total_items)
```

```
## [1] 72
```

```
## The user id of the customer who bought most items
filter(number_of_items_by_customer, total_items == 72)
```

```
## # A tibble: 1 x 2
##   user_id total_items
##     <int>       <dbl>
## 1  269335          72
```

The customer who bought the maximum items in her lifetime has the user id 269335.

Finding for each item, the customer who bought that product the most

```
## Data giving how many times each user bought each item in all transactions
d <- data_with_users %>% group_by(user_id) %>% summarise_all(funs(sum))

## Creating data to show the customer who buys that item most number of times
most_buying_customer_for_each_item <- data.frame(matrix(nrow = 48, ncol = 81))

## Renaming columns
```

```r
colnames(most_buying_customer_for_each_item) <- c("Item_name", paste0( "user_id_", 1:80))

## 1st column as the name of the item
most_buying_customer_for_each_item$Item_name <- item_to_id$Item_name

for(i in 2:ncol(d)) {
  z <- max(d[,i])
  a <- filter(d, d[,i] == z)
  for(j in 1:nrow(a)) {
    most_buying_customer_for_each_item[[(i-1), (j+1)]] = a[[j, 1]]
  }
}

x <- c()

## All the users (they might be repeated)
for(i in 1:48) {
  for(j in 2:81) {
    y <- most_buying_customer_for_each_item[[i, j]]
    if (is.na(y) == TRUE) {
      x <- x
    } else {
      x <- c(x, y)
    }
  }
}

k <- c()

## Unique users
for(i in 1:length(x)) {
  if(x[i] %in% k == FALSE) {
    k <- c(k, x[i])
  } else {
    k <- k
  }
}

## Print all users
k
```

```
##   [1]   31625   68836  540483 1091637 1301034  269335  154960  593439
##   [9] 1147269 1433188    5289   73071  432842  217277  397623  414416
##  [17] 1392068  334664 1151741  175865  312711  360336  811299 1147990
##  [25] 1494252  151926  238761  269836  297980  300878  423287  478446
##  [33]  489063  578216  587316  722795  723012  765161  851688  914267
##  [41]  973683 1054361 1119944 1168773 1238470 1264074 1274438 1374100
##  [49] 1419565 1451339 1485538 1271258 1310896  618914  743501  367872
##  [57]  534745 1038694 1198106 1249050 1435298  557904  791038  653800
##  [65]  820788  172120  255458  279962  318112  380900  384935  395775
##  [73]  490181  544364  554479  718218  764759  884172  951844  993496
##  [81] 1054816 1091106 1227423  143741   90642  189005  319296  491729
##  [89]  545108  745575  837807  888933  920036 1064792 1169085 1374867
```

```
##  [97] 1406663 1464442  366155  463073 1089642 1275324  917199 1393126
## [105]  885474 1100981 1433799 1199670  920002  189913 1077463 1121617
## [113] 1146129   68282  109578  910391 1027296 1414621  967573 1341188
## [121]  956666  204624  238495  394348   21779   48313   50451   64998
## [129]   67283   80215   88276   94543  122129  144516  146799  163459
## [137]  164141  192248  218574  220532  222206  222276  250937  269376
## [145]  293648  316538  327140  375895  380517  393183  398629  421126
## [153]  432935  454041  461033  482712  488054  517171  545543  554524
## [161]  567478  599172  605213  644456  660207  663423  681639  705714
## [169]  745944  755183  786951  806978  815473  832285  904192  912053
## [177]  912956  913744  942889  964963  968345  982566 1015177 1020422
## [185] 1021134 1027420 1049112 1068569 1076958 1153940 1157871 1222963
## [193] 1267665 1273957 1352666 1376364 1399646 1402451 1442685 1449970
## [201]  289360 1303742 1310207 1425746  305916  375849  557099 1158937
## [209]  450482 1003550 1380205  602347   46757  198866  364868  255546
## [217]  889814   38872   87247 1217810 1236029 1493728  133355  635240
## [225]  761520  776603  996380  250777  268767  297185 1286028 1218645
## [233] 1269111 1303056  335841  342220  608263  728584  943163 1167089
## [241] 1213479 1280108 1329628
```