ML-Clustering

Pratik Gandhi

July 6, 2016

Loading the data:

```
library(clValid)
```

```
## Warning: package 'clValid' was built under R version 3.2.5
## Loading required package: cluster
```

```
# setwd('C:/Users/pkgandhi/Downloads')
setwd("C:/Users/Pratik Gandhi/Documents/Data Science Stuff/Projects/MachineLearning/Clustering")
# setwd('Y:/ML_Stuff')
customers <- read.csv(file = "Wholesale customers data.csv", header = TRUE)
```

Inspecting the data:

```
# Dimensions of the data:
dim(customers)
```

```
## [1] 440   8
```

```
# Looking at the head:
head(customers, n = 5)
```

```
##   Channel Region Fresh Milk Grocery Frozen Detergents_Paper Delicassen
## 1       2      3 12669 9656    7561    214             2674       1338
## 2       2      3  7057 9810    9568   1762             3293       1776
## 3       2      3  6353 8808    7684   2405             3516       7844
## 4       1      3 13265 1196    4221   6404              507       1788
## 5       2      3 22615 5410    7198   3915             1777       5185
```

```
# Looking at the tail:
tail(customers)
```

```
##     Channel Region Fresh  Milk Grocery Frozen Detergents_Paper Delicassen
## 435       1      3 16731  3922    7994    688             2371        838
## 436       1      3 29703 12051   16027  13135              182       2204
## 437       1      3 39228  1431     764   4510               93       2346
## 438       2      3 14531 15488   30243    437            14841       1867
## 439       1      3 10290  1981    2232   1038              168       2125
## 440       1      3  2787  1698    2510     65              477         52
```

```
# Data type:
str(customers)
```

```
## 'data.frame':    440 obs. of  8 variables:
##  $ Channel         : int  2 2 2 1 2 2 2 2 1 2 ...
##  $ Region          : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ Fresh           : int  12669 7057 6353 13265 22615 9413 12126 7579 5963 6006 ...
##  $ Milk            : int  9656 9810 8808 1196 5410 8259 3199 4956 3648 11093 ...
```

```
##  $ Grocery         : int  7561 9568 7684 4221 7198 5126 6975 9426 6192 18881 ...
##  $ Frozen          : int  214 1762 2405 6404 3915 666 480 1669 425 1159 ...
##  $ Detergents_Paper: int  2674 3293 3516 507 1777 1795 3140 3321 1716 7425 ...
##  $ Delicassen      : int  1338 1776 7844 1788 5185 1451 545 2566 750 2098 ...
```

```
# Basic Intuition/statistics:
summary(customers)
```

```
##     Channel          Region          Fresh             Milk
##  Min.   :1.000   Min.   :1.000   Min.   :     3   Min.   :   55
##  1st Qu.:1.000   1st Qu.:2.000   1st Qu.:  3128   1st Qu.: 1533
##  Median :1.000   Median :3.000   Median :  8504   Median : 3627
##  Mean   :1.323   Mean   :2.543   Mean   : 12000   Mean   : 5796
##  3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.: 16934   3rd Qu.: 7190
##  Max.   :2.000   Max.   :3.000   Max.   :112151   Max.   :73498
##     Grocery          Frozen         Detergents_Paper    Delicassen
##  Min.   :    3   Min.   :   25.0   Min.   :    3.0   Min.   :    3.0
##  1st Qu.: 2153   1st Qu.:  742.2   1st Qu.:  256.8   1st Qu.:  408.2
##  Median : 4756   Median : 1526.0   Median :  816.5   Median :  965.5
##  Mean   : 7951   Mean   : 3071.9   Mean   : 2881.5   Mean   : 1524.9
##  3rd Qu.:10656   3rd Qu.: 3554.2   3rd Qu.: 3922.0   3rd Qu.: 1820.2
##  Max.   :92780   Max.   :60869.0   Max.   :40827.0   Max.   :47943.0
```

1. Here the variables Channel and Region would not be useful in Clustering. So it would be in the best of interest to drop them.
2. Overall, the difference between minimum and maximum value for all the products is really high which makes sense.
3. Scaling would be less helpful and log transformation would be a better choice here! Also we can remove the top 5 customers from each category as most of the time the folks in the middle 50% are generally targeted.

Writing a function here to eradicate the top 5 customers

Removing the top 5 customers:

```
# Function to remove n customers:

top_five_cust <- function(data, cols, n) {

    idx_to_remove <- integer(0)  # Initializing a vector of indexes to be removed

    for (c in cols) {
        # This will take the number of columns that are specified
        col_order <- order(customers[, c], decreasing = T)  # Ordering for 'c' column is done in decrea
        idx <- head(col_order, n)  # This will take index of top 'n' customers.
        idx_to_remove <- union(idx_to_remove, idx)  # Union/Intersection of the row-indexes to be remov
    }

    return(idx_to_remove)  # Returning the indexes of customers to be removed.
}

# Running the function on our data:
total_cust <- top_five_cust(customers, cols = 3:8, n = 5)
```

```r
# Looking at the customers:
customers[total_cust, ]
```

```
##     Channel Region  Fresh  Milk Grocery Frozen Detergents_Paper Delicassen
## 182       1      3 112151 29627   18148  16745             4948       8550
## 126       1      3  76237  3473    7102  16538              778        918
## 285       1      3  68951  4411   12609   8692              751       2406
## 40        1      3  56159   555     902  10002              212       2916
## 259       1      1  56083  4563    2124   6422              730       3321
## 87        2      3  22925 73498   32114    987            20070        903
## 48        2      3  44466 54259   55571   7782            24171       6465
## 86        2      3  16117 46197   92780   1026            40827       2944
## 184       1      3  36847 43950   20170  36534              239      47943
## 62        2      3  35942 38369   59598   3254            26701       2017
## 334       2      2   8565  4980   67298    131            38102       1215
## 66        2      3     85 20959   45828     36            24231       1423
## 326       1      2  32717 16784   13626  60869             1272       5609
## 94        1      3  11314  3090    2062  35009               71       2698
## 197       1      1  30624  7209    4897  18711              763       2876
## 104       1      3  56082  3504    8906  18028             1480       2498
## 24        2      3  26373 36423   22019   5154             4337      16523
## 72        1      3  18291  1266   21042   5373             4173      14472
## 88        1      3  43265  5025    8117   6312             1579      14351
```

```r
# Removing those customers and also the 'Channel' and 'Region' variables:
filter_customers <- customers[-c(total_cust), -c(1, 2)]
```

Running K-means Clustering on the filtered data:

```r
set.seed(385485)


customers_km <- kmeans(filter_customers, 5, nstart = 20)
# Here we choose 5 parameters just to test. If the number of centroids are not
# specified R will randomly assign them.  The last parameter is the number of
# times telling R to restart with different centroids


# Now WSS keeps decreasing as k increases. This results in cluster of each
# objects.

wss <- customers_km$tot.withinss
bss <- customers_km$betweenss
tss <- wss + bss

# Checking the ratio of WSS/TSS. Good ratio should be ideally less than 0.2
wss/tss
```

```
## [1] 0.2649333
```

```r
# Looking at the centers
customers_km$centers
```

```
##       Fresh      Milk   Grocery    Frozen Detergents_Paper Delicassen
```

```
## 1  4191.141  7664.294 11689.859 1287.541        5094.4000  1383.9765
## 2  5836.321  2410.226  2905.679 2766.952         686.0060   857.4226
## 3 35648.781  4787.719  5745.875 4027.312         976.0938  1563.0938
## 4 18503.162  3404.828  4552.727 3170.343        1072.6667  1444.8687
## 5  5881.459 16366.135 23980.243 2081.568       10451.8919  2058.0811
```

```r
# Cluster:
table(customers_km$cluster)
```

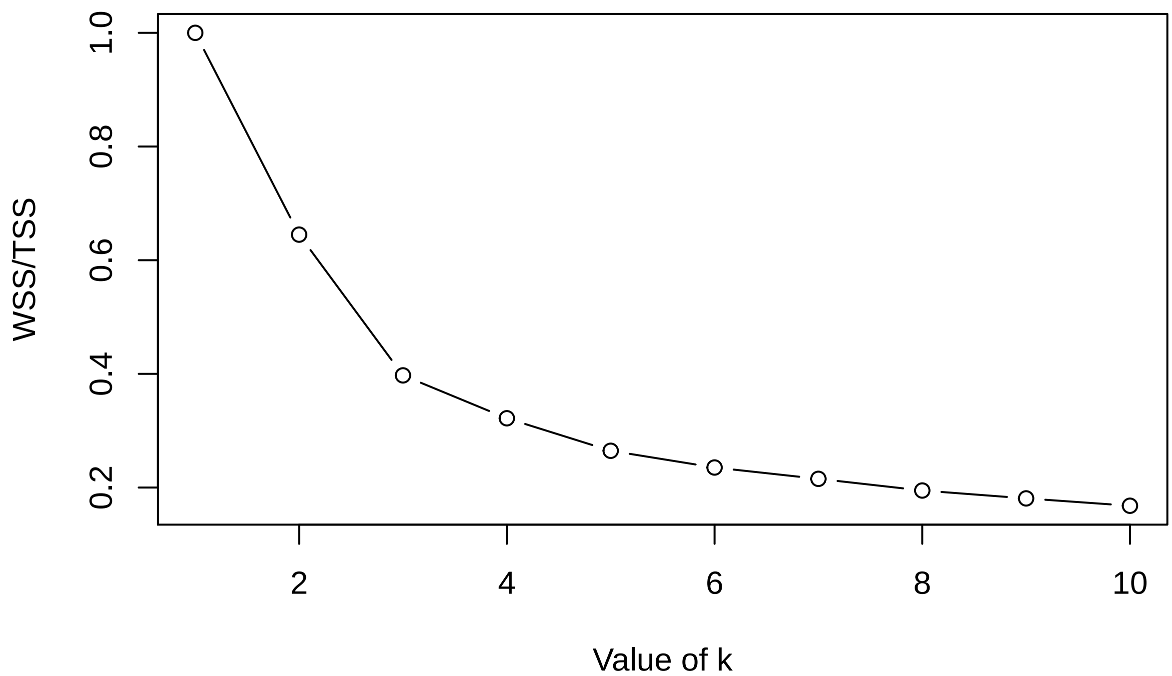
```
##
##   1   2   3   4   5
##  85 168  32  99  37
```

- We can see here that chosing 5 as number of centroids was pretty close in accordance to the ideal ratio.
- We can see that Cluster-1 is people with heavy Grocery and low Fresh foods.
- Cluster-2 represents more like low customers
- Cluster-3 is more heavy with Fresh and Frozen foods.
- Cluster-5 is more heavy with Grocery, Delicassen, Milk as well as Detergents_Paper. This looks like in the upper half of the middle 50% or may be above that!

1. We want to chose k such that clusters are compact and well separated.
2. As mentioned earlier, we do not want to end up in the situation where we have cluster for each of them because the ratio keeps on decreasing as k increases.
3. Thus we need to choose optimal k such that the ratio of WSS/TSS is insignificant.

Trying for different k values:

```r
# Initializing the variable to store multiple values:
ratio_ss <- rep(0, 10)

# For loop
for (k in 1:10) {

    # Applying the k-means:
    customers_loop_km <- kmeans(filter_customers, k, nstart = 20)

    # Saving the ratio WSS/TSS for each kth element in ratio_ss
    ratio_ss[k] <- customers_loop_km$tot.withinss/customers_loop_km$totss
}

# Making a scree plot:
plot(ratio_ss, type = "b", xlab = "Value of k", ylab = "WSS/TSS")
```



- Here we calculated ten unique ratios(WSS/TSS) for different number of centroids respectively.
- Based on the scree plot, we can definitely see that the ratio keeps on decreasing until we choose 6 as the number of centroids.
- We cannot see any tremendous amount of downfall in the ratio as we look at centroids greater than 6!

Hierarchial Clustering:

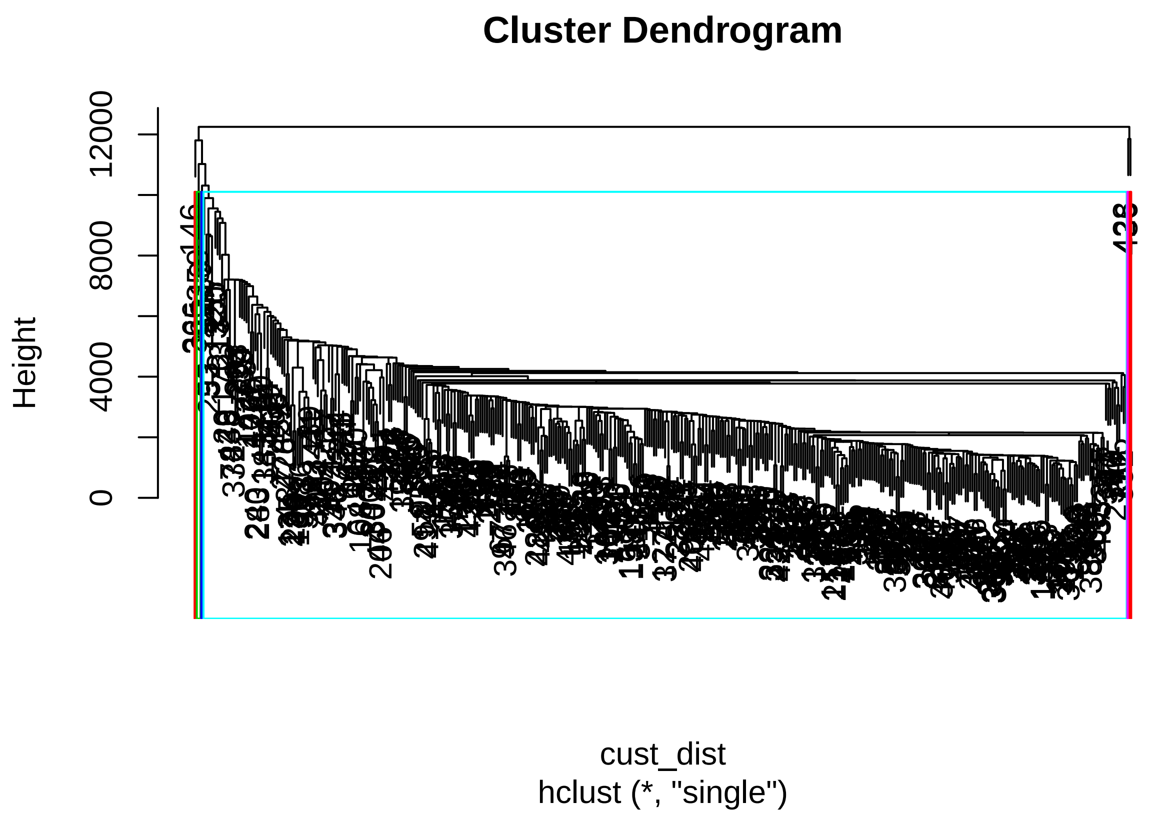
- To know more insights about the clustering:

1. Which objects cluster first?
2. Which cluster pairs merge? When?

Single Linkage Code: Minimal distance between clusters

```r
# Calculation of distance matrix of filter_customers.Here dist() used Euclidean
# method by default.
cust_dist <- dist(filter_customers)

# Clustering the data based on single-linkage method:
cust_single <- hclust(cust_dist, method = "single")

# Cutting the tree:
memb_single <- cutree(cust_single, 6)
```

```r
# Drawing the dendogram:
plot(cust_single)

# Drawing boxes around clusters using different colors:
rect.hclust(cust_single, 6, border = 2:6)
```

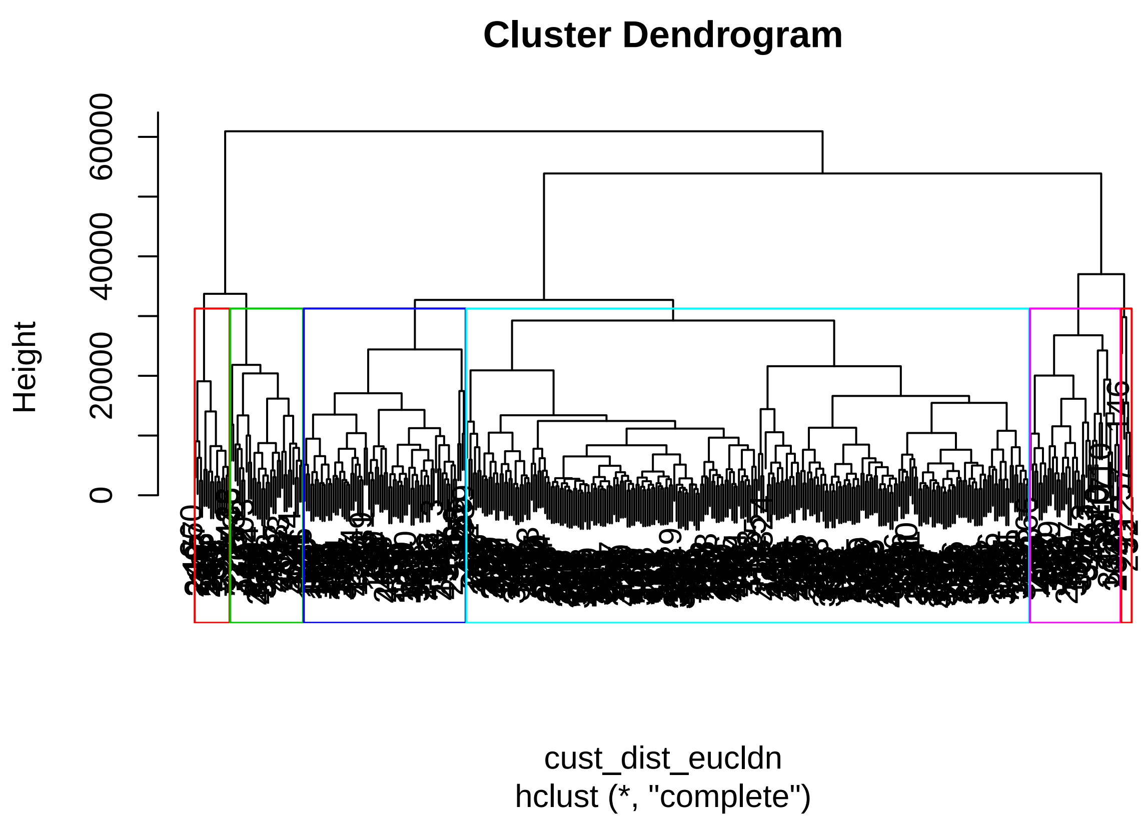


* Can be great outlier detector because the data points that are far away would merge last in cluster. * Dendogram here shows which cluster merge at which point.

Complete Linkage: Maximal distance between clusters

```r
# Calculation of distance matrix of filter_customers.
cust_dist_eucldn <- dist(filter_customers, method = "euclidean")

# Clustering the data based on complete-linkage method:
cust_complete <- hclust(cust_dist_eucldn, method = "complete")

# Cutting the tree:
memb_complete <- cutree(cust_complete, 6)

# Drawing the dendogram:
plot(cust_complete)
# plot(filter_customers, col=memb_complete)

# Drawing boxes around clusters using different colors:
```

```
rect.hclust(cust_complete, k = 6, border = 2:6)
```




```
# Comparing the membership between single and complete linkage clusterings:
table(memb_single, memb_complete)
```

```
##              memb_complete
## memb_single   1   2   3   4   5   6
##           1  70 253  31  41  16   4
##           2   0   0   0   0   0   1
##           3   2   0   0   0   0   0
##           4   1   0   0   0   0   0
##           5   0   0   1   0   0   0
##           6   0   0   1   0   0   0
```

- The five clusters differ significantly from the single-linkage clusters.
- The one big cluster we had before is now split into several clsuters.

Comparing k-means v/s Hierarchial :

```
# Dunn's index for k-means: dunn_cust_km
dunn_cust_km <- dunn(clusters = customers_km$cluster, Data = filter_customers)

# Dunn's index for single-linkage: dunn_cust_single
dunn_cust_single <- dunn(clusters = memb_single, Data = filter_customers)
```

```r
# Dunn's index for single-linkage: dunn_cust_complete
dunn_cust_complete <- dunn(clusters = memb_complete, Data = filter_customers)

# Compare k-means with single-linkage
table(customers_km$cluster, memb_single)
```

```
##    memb_single
##       1   2   3   4   5   6
##   1  84   0   0   1   0   0
##   2 168   0   0   0   0   0
##   3  30   0   0   0   1   1
##   4  99   0   0   0   0   0
##   5  34   1   2   0   0   0
```

```r
# Compare k-means with complete-linkage
table(customers_km$cluster, memb_complete)
```

```
##    memb_complete
##       1   2   3   4   5   6
##   1  66   8   0  11   0   0
##   2   4 164   0   0   0   0
##   3   0   0  16   0  16   0
##   4   1  81  17   0   0   0
##   5   2   0   0  30   0   5
```

- The table shows that the clusters obtained from the complete linkage method are similar to those of k-means.
- However the dunn's index of single linkage method is high as compare to k-means and complete linkage method.