K-means Clustering

Dataset

Wholesale Customer dataset contains data about clients of a wholesale distributor. It includes the annual spending in monetary units (m.u.) on diverse product categories. The dataset is available from the UCI ML Repository.

The dataset used in this script is partially preprocessed, where Channel and Region attributes are factorized and outliers were removed for some variables.

The objective is to segment (cluster) customers.

Let's load the dataset.

```r
customers.data <- read.csv(file = "wholesale_customers_data1.csv")
str(customers.data)
```

```r
## 'data.frame': 440 obs. of 8 variables:
##  $ Channel         : Factor w/ 2 levels "Horeca","Retail": 2 2 2 1 2 2 2 1 2 ...  
##  $ Region          : Factor w/ 3 levels "Lisbon","Oporto",..: 3 3 3 3 3 3 3 3 3 3 ...  
##  $ Fresh           : num 12669 7057 6353 13265 22615 ...  
##  $ Milk            : num 9656 9810 8808 1196 5410 ...  
##  $ Grocery         : int 7561 9568 7684 4221 7198 5126 6975 9426 6192 188  
##  $ Frozen          : num 214 1762 2405 6404 3915 ...  
##  $ Detergents_Paper: num 2674 3293 3516 507 1777 ...  
##  $ Delicatessen    : num 1338 1776 3456 1788 3456 ...  
```

Examining and Preparing the Data

We will first check if there are observations with missing values. If missing values are present, those instances should be either removed or imputed (imputation is the process of replacing missing data with substituted values).

```r
which(complete.cases(customers.data)==F)
```

```r
## integer(0)
```

In our dataset, there are no missing values.

Since the algorithm is based on measuring distances (e.g. Euclidean), this implies that all variables must be continuous and the approach can be severely affected by outliers. So, we
should check if outliers are present. We will check only numerical variables that will be used for clustering.

Box-plots are useful for detection of outliers. More details on how to analyze the box plot can be found here

```r
library(ggplot2)
ggplot(customers.data, aes(x=Channel, y=Grocery, fill=Channel)) + geom_boxplot()
```

![Box-plot for Grocery vs Channel]

```r
ggplot(customers.data, aes(x=Channel, y=Milk, fill=Channel)) + geom_boxplot()
```
ggplot(customers.data, aes(x=Channel, y=Delicatessen, fill=Channel)) + geom_boxplot()
ggplot(customers.data, aes(x=Channel, y=Frozen, fill=Channel)) + geom_boxplot()
```r
ggplot(customers.data, aes(x=Channel, y=Fresh, fill=Channel)) + geom_boxplot()
```
`ggplot(customers.data, aes(x=Channel, y=Detergents_Paper, fill=Channel)) + geom_boxplot()`
It seems that only 2 variables have outliers.

The plots also suggest that there is a considerable difference between the two distribution channels, so it would be better to examine and cluster each of them separately.

```r
retail.data <- subset(customers.data, Channel == 'Retail')
horeca.data <- subset(customers.data, Channel == 'Horeca')
```

Let's focus first on the retail.data. **For homework:** do the same with the horeca.data.

```r
summary(retail.data)
```

<table>
<thead>
<tr>
<th></th>
<th>Channel</th>
<th>Region</th>
<th>Fresh</th>
<th>Milk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Horeca: 0</td>
<td>Lisbon</td>
<td>Min.</td>
<td>Min.</td>
</tr>
<tr>
<td></td>
<td>Retail:142</td>
<td>Oporto</td>
<td>1st Qu.</td>
<td>1st Qu.</td>
</tr>
<tr>
<td></td>
<td>Other_region:105</td>
<td>Other_region:105</td>
<td>Median</td>
<td>Median</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mean</td>
<td>Mean</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3rd Qu.</td>
<td>3rd Qu.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Max.</td>
<td>Max.</td>
</tr>
<tr>
<td></td>
<td>Grocery</td>
<td>Frozen</td>
<td>Detergents_Paper</td>
<td>Delicatessen</td>
</tr>
<tr>
<td>Min.</td>
<td>2743</td>
<td>33.0</td>
<td>Min.</td>
<td>332</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>9245</td>
<td>534.2</td>
<td>1st Qu.</td>
<td>3684</td>
</tr>
<tr>
<td>Median</td>
<td>12390</td>
<td>1081.0</td>
<td>Median</td>
<td>5614</td>
</tr>
<tr>
<td>Mean</td>
<td>16323</td>
<td>1652.6</td>
<td>Mean</td>
<td>6650</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Remove the Channel variable as we now have just one channel

```
retail.data <- retail.data[, -1]
```

Check which variables have outliers

```
apply(X = retail.data[, -1], # all variables except Region
      MARGIN = 2,
      FUN = function(x) length(boxplot.stats(x)$out))
```

<table>
<thead>
<tr>
<th></th>
<th>Fresh</th>
<th>Milk</th>
<th>Grocery</th>
<th>Frozen</th>
</tr>
</thead>
<tbody>
<tr>
<td>##</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>Detergents_Paper</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delicatessen</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

So, Grocery and Frozen variables have outliers that we need to deal with.

As a way of dealing with outliers, we'll use the Winsorizing technique. Practically, it consists of replacing extreme values with a specific percentile of the data, typically 90th or 95th.

Let's start with the Grocery variable.

We will extract the outliers and sort them by their values.

```
sort(boxplot.stats(retail.data$Grocery)$out)
```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>39694</td>
<td>45828</td>
<td>55571</td>
<td>59598</td>
</tr>
</tbody>
</table>

Now, we examine the 90th, 95th, ... percentile.

```
quantile(retail.data$Grocery, probs = seq(from = 0.9, to = 1, by = 0.025))
```

<table>
<thead>
<tr>
<th></th>
<th>90%</th>
<th>92.5%</th>
<th>95%</th>
<th>97.5%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>28373.00</td>
<td>31004.18</td>
<td>34731.70</td>
<td>50455.93</td>
<td>92780.00</td>
</tr>
</tbody>
</table>

The 95th percentile seems to be a good cutting point.

```
new.max <- as.numeric(quantile(retail.data$Grocery, probs = 0.95))
retail.data$Grocery[retail.data$Grocery > new.max] <- new.max
```

By drawing the box plot again we will see that there are no outliers present any more.

```
boxplot(retail.data$Grocery, xlab='Grocery')
```
Now, we'll deal with outliers for the Frozen variable.

```r
sort(boxplot.stats(retail.data$Frozen)$out)
## [1]  4736  5154  5612  5641  6746  7782  8132  8321 11559
quantile(retail.data$Frozen, probs = c(seq(0.9, 1, 0.025)))
##  90%  92.5%   95%  97.5%  100%
## 3519.50 4258.55 5133.10 7238.10 11559.00
```

Setting values to the 92.5th percentile seems to be a good approach.

```r
new.max <- as.numeric(quantile(retail.data$Frozen, probs = 0.925))
retail.data$Frozen[retail.data$Frozen > new.max] <- new.max
boxplot(retail.data$Frozen, xlab='Frozen')
```
Finally, examine the retail.data after the transformations

```r
summary(retail.data)
```

<table>
<thead>
<tr>
<th></th>
<th>Region</th>
<th>Fresh</th>
<th>Milk</th>
<th>Grocery</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lisbon : 18</td>
<td>Min. : 18</td>
<td>Min. : 928</td>
<td>Min. : 2743</td>
</tr>
<tr>
<td></td>
<td>Oporto : 19</td>
<td>1st Qu.: 2348</td>
<td>1st Qu.: 5938</td>
<td>1st Qu.: 9245</td>
</tr>
<tr>
<td></td>
<td>Other_region:105</td>
<td>Median : 5994</td>
<td>Median : 7812</td>
<td>Median : 12390</td>
</tr>
<tr>
<td></td>
<td>Mean : 8460</td>
<td>Mean : 9421</td>
<td>Mean : 15237</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3rd Qu.:12230</td>
<td>3rd Qu.:12163</td>
<td>3rd Qu.:20184</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max. :26287</td>
<td>Max. :20638</td>
<td>Max. :34732</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Frozen</th>
<th>Detergents_Paper</th>
<th>Delicatessen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min. : 33.0</td>
<td>Min. : 332</td>
<td>Min. : 3.0</td>
</tr>
<tr>
<td></td>
<td>1st Qu.: 534.2</td>
<td>1st Qu.: 3684</td>
<td>1st Qu.: 566.8</td>
</tr>
<tr>
<td></td>
<td>Median :1081.0</td>
<td>Median : 5614</td>
<td>Median :1350.0</td>
</tr>
<tr>
<td></td>
<td>Mean :1471.9</td>
<td>Mean : 6650</td>
<td>Mean :1485.2</td>
</tr>
<tr>
<td></td>
<td>3rd Qu.:2146.8</td>
<td>3rd Qu.:8662</td>
<td>3rd Qu.:2156.0</td>
</tr>
<tr>
<td></td>
<td>Max. :4258.6</td>
<td>Max. :16171</td>
<td>Max. :3455.6</td>
</tr>
</tbody>
</table>
Clustering with 2 Features

The K-means algorithm groups all observations into K different clusters. K is given as an input parameter to the algorithm.

The algorithm works as follows: 1. Select K centroids (K observations chosen at random) 2. For each observation find the nearest centroid (based on the Euclidean or some other distance measure) and assign that observation to the centroid's cluster 3. For each cluster's centroid recalculate its position based on the mean of observations assigned to that cluster 4. Assign observations to their closest centroids (newly computed) 5. Continue with steps 3 and 4 until the observations are not reassigned or the maximum number of iterations (R uses 10 as a default) is reached.

Let's choose only two features for this initial, simple clustering.

```
pairs(~Fresh+Frozen+Grocery+Milk+Delicatessen+Detergents_Paper, data = retail.data)
```

By looking at the plots, no particular pattern can be observed. But for the sake of an example, let's pick Frozen and Milk variables.

Get a closer look at selected variables.
No clear pattern in the data, but let’s run K-means and see if some clusters will emerge.

Create a subset of the original data containing the attributes to be used in the K-means.

```
retail.data1 <- retail.data[, c("Frozen", "Milk")]
summary(retail.data1)
```

```
#> Frozen         Milk
#>  Min.   : 33.0 Min.   : 928
#>  1st Qu.:534.2 1st Qu.: 5938
#>  Median :1081.0 Median : 7812
#>  Mean   :1471.9 Mean   : 9421
#>  3rd Qu.:2146.8 3rd Qu.:12163
#>  Max.   :4258.6 Max.   :20638
```

When variables are in incomparable units and/or the numeric values are on very different scales of magnitude, they should be rescaled. Since our variables Frozen and Milk have different value ranges, we need to rescale the data. To that end, we will use normalization as there are no outliers. Normalization is done using the formula: \((x - \text{min}(x))/(\text{max}(x) - \text{min}(x))\)
We will load the file with our utility functions (code is given at the end of the script) and call our custom function for normalization.

```r
source("Utility.R")
retail.data1.norm <- as.data.frame(apply(retail.data1, 2, normalize.feature))
summary(retail.data1.norm)
```

```
##      Frozen            Milk
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.1186   1st Qu.:0.2542
##  Median :0.2480   Median :0.3493
##  Mean   :0.3405   Mean   :0.4309
##  3rd Qu.:0.5002   3rd Qu.:0.5700
##  Max.   :1.0000   Max.   :1.0000
```

Run the K Means algorithm, specifying, for example, 4 centers. 'iter.max' defines the maximum number of iterations. This overcomes the problem of situations where there is a slow convergence. This clustering approach can be sensitive to the initial selection of centroids. 'nstart' option attempts multiple initial configurations and reports on the best one. Afterwards, we inspect the results.

```r
# set the seed to guarantee that the results are reproducible.
set.seed(3108)
simple.4k <- kmeans(x = retail.data1.norm, centers=4, iter.max=20, nstart=100)
simple.4k
```

```
## K-means clustering with 4 clusters of sizes 10, 35, 72, 25
##
## Cluster means:
## # Frozen   Milk
## # 1 0.8887541 0.8903003
## # 2 0.2417335 0.7118702
## # 3 0.1734192 0.2642975
## # 4 0.7407727 0.3335116
##
## Clustering vector:  
## # 1 2 3 5 6 7 8 10 11 12 13 14 15 17 19 21 24 25
## # 2 3 2 2 3 2 3 2 3 2 1 3 3 3 3 1 3
## # 3 62 63 64 66 68 74 75 78 82 83 85 86 87 93 95 97 101 102
## # 4 1 4 2 3 4 3 2 3 3 3 2 2 1 2 3 4 2
## # 103 107 108 109 110 112 124 128 146 156 157 159 160 161 164 165 166 167
## # 4 3 4 3 2 2 4 3 3 3 3 3 3 2 4 1 3
## # 171 172 174 176 189 198 200 202 206 208 210 212 215 217 219 221 224
## # 3 2 3 3 4 2 3 3 1 1 2 2 4 2 1 3 3
## # 227 231 246 252 265 267 269 280 282 294 296 298 299 301 302 303 304 305
## # 3 4 3 1 3 4 3 3 3 2 3 3 3 3 3 2 3 3
## # 306 307 310 313 316 320 332 334 335 336 341 342 344 347 348 350 352 354
```
From the output, we can observe the following evaluation metrics:

- **within_SS (within cluster sum of squares):** the sum of squared differences between individual data points in a cluster and the cluster center (centroid); it is computed for each cluster.

- **total_SS:*** the sum of squared differences of each data point to the global sample mean.

- **between_SS:*** the sum of squared differences of each centroid to the global sample mean (when computing this value, the squared difference of each cluster center to the global sample mean is multiplied by the number of data points in that cluster).

- **between_SS / total_SS:** this ratio indicates how ‘well’ the sample splits into clusters; the higher the ratio, the better clustering. The maximum is 1.

Add the vector of clusters to the data frame as a new variable 'cluster' and plot it.

```r
retail.data1.norm$cluster <- factor(simple.4k$cluster)
head(retail.data1.norm)
```

```none
#       Frozen    Milk cluster
# 1  0.04283466 0.4428231       3
# 2  0.40917750 0.4506365       3
# 3  0.56134794 0.3997991       4
# 5  0.91869697 0.2273984       4
# 6  0.14980298 0.3719451       3
# 7  0.10578505 0.1152213       3
```

```r
# color the points by their respective clusters
gg2 <- ggplot(data=retail.data1.norm, aes(x=Frozen, y=Milk, colour=cluster)) +
   geom_point() +
   xlab("Annual spending on frozen products") +
   ylab("Annual spending on dairy products") +
   ggtitle("Retail customers annual spending")
# add cluster centers
```

```r
gg2 + geom_point(data=as.data.frame(simple.4k$centers), colour="black", size=4, shape=17)
```
The plot allows us to visually inspect the clusters, but it is better to have a more systematic approach to judging the quality of clustering and selecting the best value for K.

**Selecting the Best Value for K**

**Selection based on the Elbow Method**

Instead of guessing the correct value for K, we can take a more systematic approach to choosing the 'right' K. It is called the Elbow method, and is based on the sum of squared differences between data points and cluster centres, that is, the sum of within_SS for all the clusters (tot.withinss).

Let us run the k-means for different values for K (from 2 to 8). We will compute the metric required for the Elbow method (tot.withinss). Along the way, we'll also compute the other metric: ratio of between_SS and total_SS.

```r
# create an empty data frame
eval.metrics.2var <- data.frame()

# remove the column with clusters
retail.data1.norm$cluster <- NULL
```
# run kmeans for all K values in the range 2:8
for (k in 2:8) {
    set.seed(3108)
    km.res <- kmeans(x=retail.data1.norm, centers=k, iter.max=20, nstart = 1000)
    # combine cluster number and the error measure, write to df
    eval.metrics.2var <- rbind(eval.metrics.2var,
        c(k, km.res$tot.withinss, km.res$betweenss/km.res$totss))
}

# update the column names
names(eval.metrics.2var) <- c("cluster", "tot.within.ss", "ratio")
eval.metrics.2var

##   cluster tot.within.ss     ratio
## 1       2     12.577765 0.4350165
## 2       3      7.981614 0.6414721
## 3       4      5.790423 0.7398987
## 4       5      4.538501 0.7961340
## 5       6      3.368527 0.8486884
## 6       7      2.815509 0.8735295
## 7       8      2.374140 0.8933555

Draw the Elbow plot

```r
ggplot(data=eval.metrics.2var, aes(x=cluster, y=tot.within.ss)) +
    geom_line() +
    ggtitle("Reduction in error for different values of K\n") +
    xlab("\nClusters") +
    ylab("Total Within Cluster Sum of Squares\n") +
    scale_x_continuous(breaks=seq(from=0, to=8, by=1))
```
It seems that K=3 or K=4 would be the best options for the number of clusters.

If it is not fully clear from the plot where we have significant decrease in the tot.within.ss value, we can compute the difference between each two subsequent values. The compute.difference is a utility function.

```r
data.frame(K=2:8, 
    tot.within.ss.delta=compute.difference(eval.metrics.2var$tot.within.ss), 
    ratio.delta=compute.difference(eval.metrics.2var$ratio))
```

##   K tot.within.ss.delta ratio.delta
## 1 2                  NA          NA
## 2 3           4.5961507  0.20645554
## 3 4           2.1911907  0.09842659
## 4 5           1.2519218  0.05623536
## 5 6           1.1699738  0.05255432
## 6 7           0.5530184  0.02484116
## 7 8           0.4413690  0.01982595

As we've already examined solution with 4 clusters, let's also examine the solution with K=3.
set.seed(3108)
simple.3k <- kmeans(x = retail.data1.norm, centers=3, iter.max=20, nstart=100)
retail.data1.norm$cluster <- factor(simple.3k$cluster)

Plot the 3-cluster solution:

```r
ggplot(data=retail.data1.norm, aes(x=Frozen, y=Milk, colour=cluster)) + geom_point() +
  xlab("Annual spending on frozen products") +
  ylab("Annual spending on dairy products") +
  ggtitle("Retail customers annual spending") +
  geom_point(data=as.data.frame(simple.3k$centers),
             colour="black", size=4, shape=17)
```

Next, we examine clusters closer by looking into the cluster centers (mean) and standard deviation from the centers. When examining clusters - in order to interpret them - we use 'regular' (not normalized) features. The summary.stats() f. used below is a custom function defined in the Utility.R.

```r
sum.stats <- summary.stats(feature.set = retail.data1,
                           clusters = simple.3k$cluster,
                           cl.num = 3)
```
### sum.stats

<table>
<thead>
<tr>
<th>attributes</th>
<th>Mean (SD)</th>
<th>Mean (SD)</th>
<th>Mean (SD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 cluster</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2 freq</td>
<td>84</td>
<td>26</td>
<td>32</td>
</tr>
<tr>
<td>3 Frozen</td>
<td>793.04 (526.94)</td>
<td>1328.08 (879.37)</td>
<td>3370.69 (797.97)</td>
</tr>
<tr>
<td>4 Milk</td>
<td>6862.69 (2782.19)</td>
<td>17342.03 (3268.35)</td>
<td>9699.39 (5204.43)</td>
</tr>
</tbody>
</table>

The solution with K=4 seems to be better as features are less dispersed than here (e.g. the Milk variable in cluster 3, and Frozen variable in cluster 2 have very high dispersion). Solution with K=4 resolves this (check the plot, and examine cluster centers).

### Clustering with All Numeric Features

Since the 6 numeric attributes differ in their value ranges, we need to normalize them, that is, transform them to the value range [0,1]

```
retail.norm <- as.data.frame(apply(retail.data[,c(2:7)], 2, normalize.feature))
summary(retail.norm)
```

Instead of guessing K, we’ll right away use the Elbow method to find the optimal value for K.

```
# create an empty data frame
eval.metrics.6var <- data.frame()

# run kmeans for all K values in the range 2:8
for (k in 2:8) {
  set.seed(3108)
  km.res <- kmeans(x=retail.norm, centers=k, iter.max=20, nstart = 1000)
  # combine cluster number and the error measure, write to df
  eval.metrics.6var <- rbind(eval.metrics.6var,}
```r
# update the column names
names(eval.metrics.6var) <- c("cluster", "tot.within.ss", "ratio")
```

```
# cluster  tot.within.ss     ratio
# 1        2      49.72071 0.2667494
# 2        3      39.96778 0.4105797
# 3        4      35.23843 0.4803252
# 4        5      31.13015 0.5409116
# 5        6      27.49447 0.5945284
# 6        7      25.18087 0.6286479
# 7        8      23.36426 0.6554381
```

```
Draw the Elbow plot

```r
ggplot(data=eval.metrics.6var, aes(x=cluster, y=tot.within.ss)) +
ggtitle("Reduction in error for different values of K\n") +
xlab("\nClusters") +
ylab("Total Within Cluster Sum of Squares\n") +
scale_x_continuous(breaks=seq(from=0, to=8, by=1))
```
This time it seems that solution with 3 clusters might be the best.

As previously done, we'll also look for the best value for K by examining the difference between each two subsequent values of tot.within.ss and ratio (of between.SS and total.SS).

```r
data.frame(k=c(2:8),
            tot.within.ss.delta=compute.difference(eval.metrics.6var$tot.within.ss),
            ration.delta=compute.difference(eval.metrics.6var$ratio))
```

<table>
<thead>
<tr>
<th></th>
<th>k</th>
<th>tot.within.ss.delta</th>
<th>ration.delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>#2</td>
<td>2</td>
<td>9.752931</td>
<td>0.14383025</td>
</tr>
<tr>
<td>#3</td>
<td>3</td>
<td>4.729349</td>
<td>0.06974554</td>
</tr>
<tr>
<td>#4</td>
<td>4</td>
<td>4.108280</td>
<td>0.06058640</td>
</tr>
<tr>
<td>#5</td>
<td>5</td>
<td>3.635680</td>
<td>0.05361679</td>
</tr>
<tr>
<td>#6</td>
<td>6</td>
<td>2.313597</td>
<td>0.03411951</td>
</tr>
<tr>
<td>#7</td>
<td>7</td>
<td>1.816606</td>
<td>0.02679019</td>
</tr>
</tbody>
</table>

This again suggests 3 clusters as the best option.

Let's examine in detail clustering with K=3.
set.seed(3108)
retail.3k.6var <- kmeans(x=retail.norm, centers=3, iter.max=20, nstart = 1000)

# examine cluster centers
sum.stats <- summary.stats(feature.set = retail.norm, #retail.data[,c(2:7)],
                           clusters = retail.3k.6var$cluster,
                           cl.num = 3)
sum.stats

##         attributes   Mean (SD)   Mean (SD)   Mean (SD)
## 1          cluster       1         2         3
## 2             freq        43        25        74
## 3            Fresh  0.55 (0.31)  0.36 (0.3) 0.18 (0.17)
## 4             Milk  0.32 (0.18)  0.83 (0.23) 0.36 (0.19)
## 5          Grocery  0.22 (0.13)  0.81 (0.18) 0.34 (0.18)
## 6           Frozen  0.5 (0.32)  0.44 (0.34) 0.21 (0.19)
## 7 Detergents_Paper  0.2 (0.12)  0.83 (0.18) 0.37 (0.19)
## 8     Delicatessen 0.63 (0.27)  0.59 (0.33) 0.26 (0.22)

Appendix: Utility Functions ('Utility.R' file)

## function that computes the difference between two consecutive values
compute.difference <- function(values) {
  dif <- vector(mode = "numeric", length = length(values))
  dif[1] <- NA
  for(i in 1:(length(values)-1)) {
    dif[i+1] <- abs(values[i+1] - values[i])
  }
  dif
}

## function that provides summary statistics about clusters
summary.stats <- function(feature.set, clusters, cl.num) {
  sum.stats <- aggregate(x = feature.set,
                         by = list(clusters),
                         FUN = function(x) {
                           m <- mean(x, na.rm = T)
                           sd <- sqrt(var(x, na.rm = T))
                           paste(round(m, digits = 2), " (",
                                  round(sd, digits = 2), ")", sep = "")
                         })
  sum.stat.df <- data.frame(cluster = sum.stats[,1],
                            freq = as.vector(table(clusters)),
                            sum.stats[,-1])
  sum.stats.transpose <- t(as.matrix(sum.stat.df)
  sum.stats.transpose <- as.data.frame(cbind(attributes, sum.stats.transpose


```r
colnames(sum.stats.transpose) <- c("attributes", rep("Mean (SD)", cl.num))
rownames(sum.stats.transpose) <- NULL
sum.stats.transpose

normalize.feature <- function(feature) {
  if (sum(feature, na.rm = T) == 0) feature
  else ((feature - min(feature, na.rm = T))/(max(feature, na.rm = T) - min(feature, na.rm = T)))
}
```