Unsupervised Algorithms in machine learning [*]

Cesar Conejo Villalobos *Data Scientist*

This document provides some examples of unsupervised algorithms in machine learning. In these techniques, we need to infer the properties of the observations without the help of an output variable or *supervisor*. We review two methods: k-means and hierarchical clustering. Then we use some data from Kaggle for applying these techniques to produce a customer segmentation. The platform that we use is R. Because of the number of observations, we are going to use a parallel process for improving the execution times.

Keywords: Unsupervised, algorithms, k-means, hierarchical classification, kaggle, R, parallel

Introduction

In the book *The Elements of Statistical Learning* Hastie T. (2008) explains that in the case of unsupervised learning, data usually has a set of N observations $(x_1, ..., x_N)$ of a random vector X having joint density $Pr(X)$. The goal is to infer the properties of this probability density.

The techniques that statistics and machine learning offer us for unsupervised learning are the following:

1) Principal components, multidimensional scaling.
2) Cluster analysis.
3) Mixture modeling.
4) Association rules.

In this exercise, we are going to focus on cluster analysis. The basis of our model will be the Kaggle Credit Card dataset for Clustering. The data are an 8950 x 18 matrix. One variable is categorical and represents the customer ID, the next seventeen are real numbers each representing the behavior of credit cardholders. The goal is to define a marketing strategy based on customer segmentation.

The first aspect we need to solve is to find the number of clusters that we need. Hastie T. (2008) says that we can have two scenarios:

1) For data segmentation, the number of clusters is defined as part of the problem, and it is base on the capacity and resources of the company. The goal is to find observations that belong to each proposed group.

2) Determine how the observations belong to natural distinct groupings. In this case, the number of clusters is unknown.

For this exercise, we are going to use scenario 2 and trying to find the number of clusters and the characteristics of each group using the following techniques:

1) k-means
2) Hierarchical clustering.

[*]Template taken from (http://github.com/svmiller). **Corresponding author**: svmille@clemson.edu.

Techniques

As mentioned before, the goal of unsupervised algorithms is to get the classes as homogeneous as possible and such that they are sufficiently separated. This goal can be specified numerically from the following property:

Suppose that exist a partition $P = (C_1, ..., C_K)$ of Ω, where $g_1, ..., g_K$ are the cluster center of the classes:

$$g_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$$

Also g is the global center $\frac{1}{N} \sum_{i=1}^{N} x_i$. We also define:

- Total point scatter: $T = \frac{1}{N} \sum_{i=1}^{N} ||x_i - g||^2$.
- Within-cluster point scatter: $W(C) = \frac{1}{N} \sum_{k=1}^{K} \sum_{i \in C_k} ||x_i - g_k||^2$.
- Between-cluster point scatter: $B(C) = \sum_{k=1}^{K} \frac{|C_k|}{N} ||g_k - g||^2$

In this case, the algorithm requires $B(C)$ to be maximum and $W(C)$ to be minimum. Since the total point scatter T is fixed, then maximizing $B(C)$ automatically minimizes $W(C)$. Therefore, the two goals (homogeneity within classes and separation between classes) are achieved at the same time by minimizing $W(C)$.

Thus, the goal in the *K-means* method is to find a partition C of Ω. Also, we find some representatives of the classes, such that $W(C)$ is minimal. For determining how many clusters a dataset has, we can use the elbow method.

Furthermore, k-means depend on the choice of the number of clusters. On the other hand, hierarchical clustering methods do not expect such designations. Instead, Hastie T. (2008) claims that this method demands the user to specify a measure of dissimilarity between (disjoint) groups of observations, based on the pairwise dissimilarities among the observations.

This method of classification uses a notion of proximity between groups of elements to measure the separation between the classes sought. To do this, the concept of aggregation is introduced, which is nothing more than a dissimilarity between groups of individuals: be A, $B \subseteq \Omega$ then the aggregation between A and B is $\delta(A, B)$. Then we have the following agglomerative clustering methods:

- Single linkage: $\delta_{SL}(A, B) = min\{d(x_i, d_j) | x_i \in A, x_j \in B\}$
- Complete linkage: $\delta_{CL}(A, B) = max\{d(x_i, d_j) | x_i \in A, x_j \in B\}$
- Average linkage: $\delta_{AL}(A, B) = \frac{1}{|A||B|} \sum_{x_i \in A, x_j \in B} d(x_i, d_j)$
- Ward linkage: $\delta_{Ward}(A, B) = \frac{|A||B|}{|A|+|B|} ||g_A - g_B||^2$

Analysis

We are going to define the marketing strategy using k-means and hierarchal clustering. But first, we will see the distribution of the data.

Exploratory Analysis

We create the function ***cc_stats()*** for analyzing some of the characteristics of the dataset such as:

- Number of complete observations.
- Number of `NA` values.
- Mean of complete observations.
- Standard desviation of complete observations.
- Number of outliers observations ($Q3 + 1.5IQR$)
- Minimun value of complete observations.
- Maximun value of complete observations.
- 95 quantile
- Upper limit for the value. (mean + 3 sd)

```
---
# Basic statistics
# Input: x vector
# Output: Summary of statistics of the input

cc_stats <- function(x){

  #NA Values
  nas = sum(is.na(x))

  # Vector with complete values
  a = x[!is.na(x)]

  # Properties

  m   = mean(a)
  min = min(a)
  max = max(a)
  s   = sd(a)

  # Stats
  stats <- boxplot.stats(a)
  n     <- stats$n
  out   <- length(stats$out)

  Q95 = quantile(a, 0.95)
  UL = m + 3*s

  return(c(n     = n,
           nas    = nas,
           Mean   = m,
           StDev  = s,
           Q_out  = out,
           Min    = min,
           Max    = max,
           Q    = Q95,
           Upper_Limit = UL))
}
```

Using the function ***apply()***, we see the statistical characteristics for each of the variables:

```
# Vector with the name of the variable
vars <- c("BALANCE",
          "BALANCE_FREQUENCY",
          "PURCHASES",
          "ONEOFF_PURCHASES",
          "INSTALLMENTS_PURCHASES",
          "CASH_ADVANCE",
          "PURCHASES_FREQUENCY",
          "ONEOFF_PURCHASES_FREQUENCY",
          "PURCHASES_INSTALLMENTS_FREQUENCY",
          "CASH_ADVANCE_FREQUENCY",
          "CASH_ADVANCE_TRX",
          "PURCHASES_TRX",
          "CREDIT_LIMIT",
          "PAYMENTS",
          "MINIMUM_PAYMENTS",
          "PRC_FULL_PAYMENT",
          "TENURE")
```

```
# Apply the function for each variable
describe_stats <- t(data.frame(apply(cc_general[vars], 2, cc_stats)))
describe_stats
```

```
##                                     n nas    Mean   StDev Q_out   Min     Max
## BALANCE                          8950   0 1564.47 2081.53   695 0.000 19043.1
## BALANCE_FREQUENCY                8950   0    0.88    0.24  1493 0.000     1.0
## PURCHASES                        8950   0 1003.20 2136.63   808 0.000 49039.6
## ONEOFF_PURCHASES                 8950   0  592.44 1659.89  1013 0.000 40761.2
## INSTALLMENTS_PURCHASES           8950   0  411.07  904.34   867 0.000 22500.0
## CASH_ADVANCE                     8950   0  978.87 2097.16  1030 0.000 47137.2
## PURCHASES_FREQUENCY              8950   0    0.49    0.40     0 0.000     1.0
## ONEOFF_PURCHASES_FREQUENCY       8950   0    0.20    0.30   782 0.000     1.0
## PURCHASES_INSTALLMENTS_FREQUENCY 8950   0    0.36    0.40     0 0.000     1.0
## CASH_ADVANCE_FREQUENCY           8950   0    0.14    0.20   525 0.000     1.5
## CASH_ADVANCE_TRX                 8950   0    3.25    6.82   804 0.000   123.0
## PURCHASES_TRX                    8950   0   14.71   24.86   766 0.000   358.0
## CREDIT_LIMIT                     8949   1 4494.45 3638.82   248 50.000 30000.0
## PAYMENTS                         8950   0 1733.14 2895.06   808  0.000 50721.5
## MINIMUM_PAYMENTS                 8637 313  864.21 2372.45   841  0.019 76406.2
## PRC_FULL_PAYMENT                 8950   0    0.15    0.29  1474  0.000     1.0
## TENURE                           8950   0   11.52    1.34  1366  6.000    12.0
##                                     Q.95% Upper_Limit
## BALANCE                           5909.11     7809.07
## BALANCE_FREQUENCY                    1.00        1.59
## PURCHASES                         3998.62     7413.11
## ONEOFF_PURCHASES                  2671.09     5572.10
## INSTALLMENTS_PURCHASES            1750.09     3124.08
## CASH_ADVANCE                      4647.17     7270.36
## PURCHASES_FREQUENCY                  1.00        1.69
## ONEOFF_PURCHASES_FREQUENCY           1.00        1.10
## PURCHASES_INSTALLMENTS_FREQUENCY     1.00        1.56
## CASH_ADVANCE_FREQUENCY               0.58        0.74
## CASH_ADVANCE_TRX                    15.00       23.72
## PURCHASES_TRX                       57.00       89.28
## CREDIT_LIMIT                     12000.00    15410.90
## PAYMENTS                          6082.09    10418.34
## MINIMUM_PAYMENTS                  2766.56     7981.55
## PRC_FULL_PAYMENT                     1.00        1.03
## TENURE                              12.00       15.53
```

First of all, there is only a few values with NA. If we want to see if they both happen at the same time, we can do:

```
sum(is.na(cc_general$CREDIT_LIMIT) & is.na(cc_general$MINIMUM_PAYMENTS))
```

```
## [1] 0
```

As a result, the NA values do not occur in the same row. For fixing these unknown values, we can follow three alternatives:

- Remove the cases.
- Fill in the unknowns using some strategy.
- Use tools that handle these types of values.

In this case, the unknown values only represent 3.51%, so we decide delete that observations.

```
cc_general <- cc_general[-which(is.na(cc_general$CREDIT_LIMIT)
                              | is.na(cc_general$MINIMUM_PAYMENTS)),]
```

Other thing we see is that the variables are measure in diferent scales. For example *BALANCE FREQUENCY*, *PURCHASES FREQUENCY*, *ONE OFF PURCHASES FREQUENCY* and *PURCHASES INSTALLMENTS FREQUENCY* are measure with a score between 0 and 1. Other values are measure in money units and others in number of transactions. Because there are different units then we should scaling variables. We do that with the function ***normalize()***:

```r
---
## Normalize
## Input: Numeric vector
## Output: Vector normalized.

normalize <- function(x){

  min_x <- min(x)
  max_x <- max(x)

  return((x - min_x)/(max_x - min_x))

}
```

Then, we apply the function to each variable

```r
cc_general_norm     <- data.frame(apply(cc_general[vars], 2, normalize))
```

Finally, we apply ***cc_stats()*** again for seeing the changes in our data.

```r
describe_stats_norm <- t(data.frame(apply(cc_general_norm[vars], 2, cc_stats)))
describe_stats_norm
```

```
##                                     n nas  Mean StDev Q_out Min Max Q.95%
## BALANCE                          8636   0 0.084 0.110   666   0   1 0.312
## BALANCE_FREQUENCY                8636   0 0.895 0.208  1511   0   1 1.000
## PURCHASES                        8636   0 0.021 0.044   767   0   1 0.083
## ONEOFF_PURCHASES                 8636   0 0.015 0.041   961   0   1 0.067
## INSTALLMENTS_PURCHASES           8636   0 0.019 0.041   811   0   1 0.080
## CASH_ADVANCE                     8636   0 0.021 0.045   976   0   1 0.100
## PURCHASES_FREQUENCY              8636   0 0.496 0.401     0   0   1 1.000
## ONEOFF_PURCHASES_FREQUENCY       8636   0 0.206 0.300   749   0   1 1.000
## PURCHASES_INSTALLMENTS_FREQUENCY 8636   0 0.369 0.398     0   0   1 1.000
## CASH_ADVANCE_FREQUENCY           8636   0 0.092 0.135   346   0   1 0.389
## CASH_ADVANCE_TRX                 8636   0 0.027 0.056   794   0   1 0.122
## PURCHASES_TRX                    8636   0 0.042 0.070   716   0   1 0.165
## CREDIT_LIMIT                     8636   0 0.149 0.122   243   0   1 0.399
## PAYMENTS                         8636   0 0.035 0.057   785   0   1 0.121
## MINIMUM_PAYMENTS                 8636   0 0.011 0.031   841   0   1 0.036
## PRC_FULL_PAYMENT                 8636   0 0.159 0.296  1343   0   1 1.000
## TENURE                           8636   0 0.922 0.218  1290   0   1 1.000
##                                  Upper_Limit
## BALANCE                                 0.41
## BALANCE_FREQUENCY                       1.52
## PURCHASES                               0.15
## ONEOFF_PURCHASES                        0.14
## INSTALLMENTS_PURCHASES                  0.14
## CASH_ADVANCE                            0.16
## PURCHASES_FREQUENCY                        1.70
## ONEOFF_PURCHASES_FREQUENCY                 1.11
## PURCHASES_INSTALLMENTS_FREQUENCY           1.56
## CASH_ADVANCE_FREQUENCY                     0.50
## CASH_ADVANCE_TRX                           0.20
## PURCHASES_TRX                              0.25
## CREDIT_LIMIT                               0.52
## PAYMENTS                                   0.21
## MINIMUM_PAYMENTS                           0.10
## PRC_FULL_PAYMENT                           1.05
## TENURE                                     1.58
```

Now, all the variables are in a scale from 0 to 1. Also, there is no change in the variance of the variables.

K-means

For applying the *k-means*, we develop the following code. We set the seed 1234 for reproducibility purposes.

```
set.seed(1234)
```

First, we need decide the number of cluster. We apply K-means clustering to the data using the following techniques:

- Hartigan-Wong
- MacQueen
- Lloyd
- Forgy

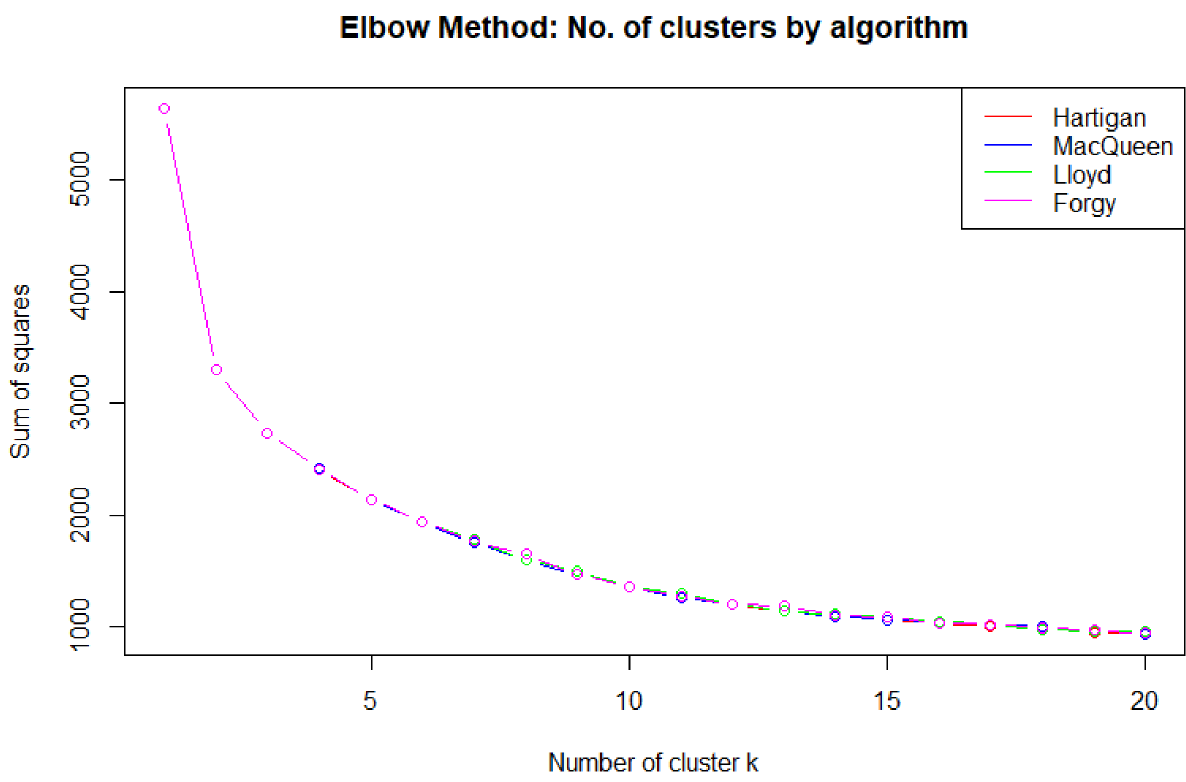
```
library(snow)

# How many k?

cl <- makeCluster(4, type="SOCK")

ignore <- clusterEvalQ(cl, {library(MASS); NULL})

#Hartigan-Wong
results_HW <- lapply(seq(1,20), function(x) kmeans(cc_general_norm,
                                                   centers = x,
                                                   algorithm = "Hartigan-Wong",
                                                   nstart = 20))

variance_HW <- sapply(results_HW, function(results_HW) results_HW$tot.withinss)
```

```r
# MacQueen
results_MQ <- lapply(seq(1,20), function(x) kmeans(cc_general_norm,
                                                   centers = x,
                                                   algorithm = "MacQueen",
                                                   nstart = 20))

variance_MQ <- sapply(results_MQ, function(results_MQ) results_MQ$tot.withinss)

# Lloyd
results_Ll <- lapply(seq(1,20), function(x) kmeans(cc_general_norm,
                                                   centers = x,
                                                   algorithm = "Lloyd",
                                                   nstart = 20))

variance_Ll <- sapply(results_Ll, function(results_Ll) results_Ll$tot.withinss)

# Forgy
results_Fg <- lapply(seq(1,20), function(x) kmeans(cc_general_norm,
                                                   centers = x,
                                                   algorithm = "Forgy",
                                                   nstart = 20))

variance_Fg <- sapply(results_Fg, function(results_Fg) results_Fg$tot.withinss)

stopCluster(cl)
```

Then, we can observe the total within-cluster sum of squares for K-means clustering for some cluster from 1 to 20.

```r
plot(variance_HW,
     col = "red",
     type = "b",
     xlab = "Number of cluster k",
     ylab = "Sum of squares",
     main = "Elbow Method: No. of clusters by algorithm")
points(variance_MQ, col = "blue", type = "b")
points(variance_Ll, col = "green", type = "b")
points(variance_Fg, col = "magenta", type = "b")
legend("topright",
       legend = c("Hartigan","MacQueen","Lloyd","Forgy"),
       col = c("red", "blue", "green", "magenta"),
       lty = 1,
       lwd = 1)
```

We can see that the kink occur at $k = 4$, so this is the number of cluster that we propose for the



marketing strategy.

Now, the question that we need to answer is which method for k-means to use. Therefore, we code the following lines that shows the results of the clustering in 4 groups.

```
# Which method:

# Hartigan-Wong
results_HW <- lapply(seq(1,20), function(x) kmeans(cc_general_norm,
                                                    centers = 4,
                                                    nstart = 20,
                                                    algorithm = "Hartigan-Wong"))
betweenss_HW <- sapply(results_HW, function(results_HW) results_HW$betweenss)

# MacQueen
results_MQ <- lapply(seq(1,20), function(x) kmeans(cc_general_norm,
                                                    centers = 4,
                                                    nstart = 20,
                                                    algorithm = "MacQueen"))
betweenss_MQ <- sapply(results_MQ, function(results_MQ) results_MQ$betweenss)


# Lloyd
results_Ll <- lapply(seq(1,20), function(x) kmeans(cc_general_norm,
                                                    centers = 4,
                                                    nstart = 20,
                                                algorithm = "Lloyd"))
betweenss_Ll <- sapply(results_Ll, function(results_Ll) results_Ll$betweenss)


# Forgy
results_Fg <- lapply(seq(1,20), function(x) kmeans(cc_general_norm,
                                                centers = 4,
                                                nstart = 20,
                                                algorithm = "Forgy"))
betweenss_Fg <- sapply(results_Fg, function(results_Fg) results_Fg$betweenss)



AVG_Between_Class <- data.frame(Method = c("Hartigan-Wong", "MacQueen",
                                         "Lloyd", "Forgy"),
                     AVG_Between_Class = c(mean(betweenss_HW), mean(betweenss_MQ),
                                         mean(betweenss_Ll),mean(betweenss_Fg))
)
```

```
Method        | Avg between
------------- | -------------
Hartigan-Wong | 3222.460
MacQueen      | 3221.499
Lloyd         | 3220.927
Forgy         | 3221.112
```

Because Lloyd reaches the minimun average between-cluster point scatter, we choose that method. As a result, we code the method in the following way:

```
Cluster_FG <- kmeans(x = cc_general_norm,
                     centers =  4,
                     iter.max = 100,
                     nstart = 20,
                     algorithm = "Lloyd")
```

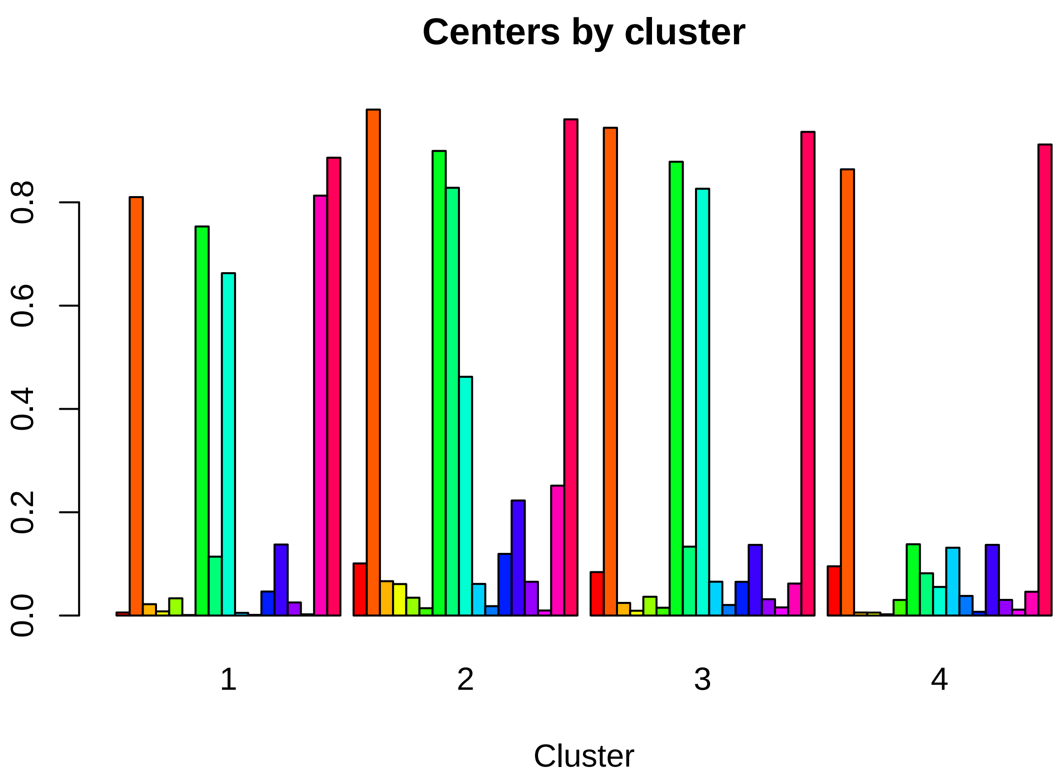
Finally, we can see some properties of the cluster. For example, the number of observations by cluster is:

```
Cluster_FG$size
```

```
## [1]  896 1249 2146 4345
```

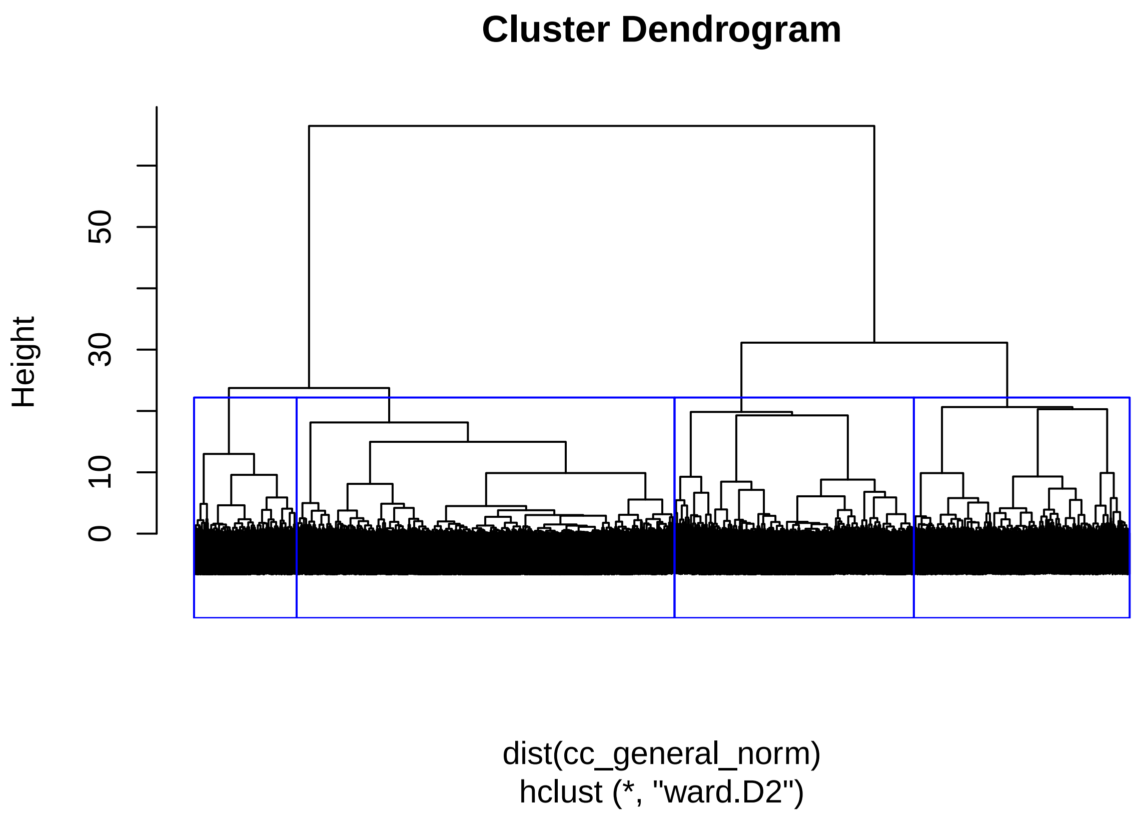
In this case, we can see that the classes are well balanced. Also, we can see the center of each cluster and realize some interpretations.

```
barplot(t(Cluster_FG$centers),
        main = "Centers by cluster",
        xlab = "Cluster",
        beside = TRUE,
        col = rainbow(17)
        )
```



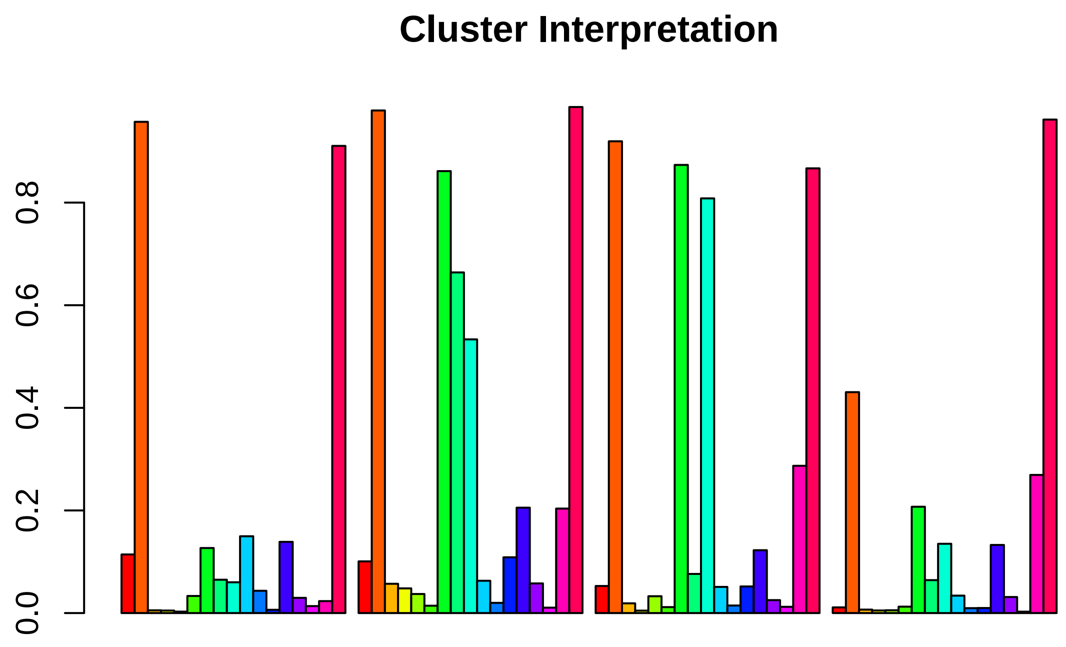
Hierachical cluster

```
model_Ward <- hclust(dist(cc_general_norm), method = "ward.D2")

plot(model_Ward, labels = FALSE)
rect.hclust(model_Ward, k = 4, border = "blue")
```



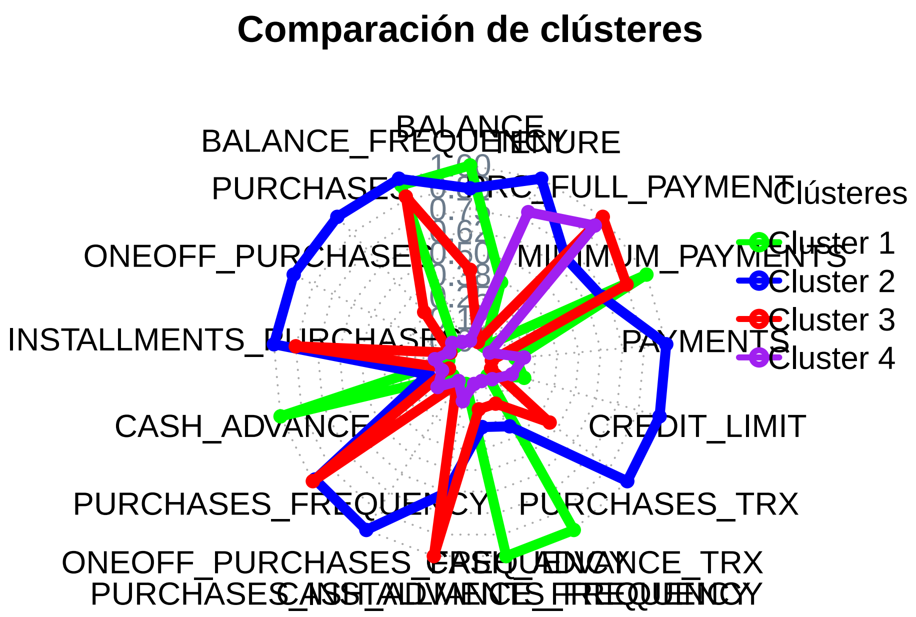
```r
cluster <- cutree(model_Ward, k = 4)

cc_general_norm_cluster <-cbind(cc_general_norm, cluster)

library(rattle)

cc_general_center <- centers.hclust(cc_general_norm,
                                    model_Ward,
                                    nclust = 4,
                                    use.median = FALSE)
```

```r
barplot(t(cc_general_center),
        beside = TRUE,
        main = "Cluster Interpretation",
        col = rainbow(17)
        )
```



```
## radar graph
center  <- as.data.frame(cc_general_center)
maximos <- apply(center,2,max)
minimos <- apply(center,2,min)
center  <- rbind(minimos,center)
center  <- rbind(maximos,center)
```

```
library(fmsb)
radarchart(center,
           maxmin = TRUE,
           axistype = 4,
           axislabcol = "slategray4",
           centerzero = FALSE,
           seg = 8,
           cglcol = "gray67",
           pcol=c("green","blue","red", "purple", "black", "brown"),
           plty = 1,
           plwd = 5,
           title = "Comparación de clústeres")



legenda <-legend(1.5,1, legend=c("Cluster 1","Cluster 2","Cluster 3", "Cluster 4"),
                 seg.len=-1.4,
                 title="Clústeres",
                 pch=21,
                 bty="n" ,lwd=3, y.intersp=1, horiz=FALSE,
                 col=c("green","blue","red", "purple", "black", "brown")
                 )
```



Conclusion

References

Hastie T., Tibshirani R., Friedman J. 2008. *The elements of Statistical Learning*. Springer.