Caso Hatco: Análisis y toma de decisiones en el entorno R

Jordi López Sintas

5 de septiembre de 2014

Acceso a los datos

Para introducir en el entorno R los datos recolectados por la dirección de Hatco tenemos varias opciones que veremos a lo largo del curso. Hoy vamos a ver cómo leer ficheros de datos generados por el programa SPSS. En el entorno R de análisis de datos disponemos del paquete `foreign` con el que podemos cómodamente acceder a datos en el formato utilizado por SPSS. Para utilizar el paquete `foreign` tenemos que cargarlo la la función library(como mostramos a continuación:

```
library(foreign)
```

Una vez cargado el paquete podemos utilizar las funciones que los ofrece. Concretamente nos interesa la función `read.spss(<nombre.del.fichero.de.datos>`. Recuerda indicar el directorio en el que se encuentra el fichero. Si lo has grabado en una memoria usb, entonces en el argumento deberás especificar la dirección completa de la localización del fichero en tu sistema informático. En el ejemplo, el fichero está en el mismo directorio que este documento:

```
hatco<-read.spss("hatconuevo.sav")
#Después de leer los datos podemos listar los datos. En el ejemplo que sigue hemos
#transpuesto la salida ordinaria con el objeot de facilitar la lectura
#con la función options() hemos limitado la anchura de las líneas de los
#informes a 58 caracteres.
options(width=58)
t(head(hatco))
```

```
##      DELSPEED    PRICELEV    PRICEFLE    MANUFIMA
## [1,] Numeric,100 Numeric,100 Numeric,100 Numeric,100
##      SERVICE     SALESFOR
## [1,] Numeric,100 Numeric,100
```

```
#Aqúi mostramos cómo leer el fichero "hatco.completo.csv" en el formato de intercambio csv.
hatco.completo <- read.csv("hatco.completo.csv", sep=";", dec=",")
#Si sólo queremos comprobar que hemos leído los datos correctamente, podemos
#utilizar la función head(<nombre.del.fichero>) que nos motrará las primeras 6
#líneas del fichero. La función tail() meustra las últimas  6 líneas.
t(head(hatco))
```

```
##      DELSPEED    PRICELEV    PRICEFLE    MANUFIMA
## [1,] Numeric,100 Numeric,100 Numeric,100 Numeric,100
##      SERVICE     SALESFOR
## [1,] Numeric,100 Numeric,100
```

```r
#Si queremos conocer únicamente el nombre de las variables que contiene el
#fichdro de datos, podemos utilizar la función names(<nombre.del.fichero>)
names(hatco)
```

```
##  [1] "DELSPEED" "PRICELEV" "PRICEFLE" "MANUFIMA"
##  [5] "SERVICE"  "SALESFOR" "PRODUCTQ" "TAMEMP"
##  [9] "USAGELEV" "SATISFLE" "ESPCOMPR" "ESTRCOMP"
## [13] "INDUSTRI" "SITCOMP"  "RANDZ"
```

```r
#La función str(<nombre.del.fichero>), en cambio, nos proporciona una breve
#descripción de las variables contenidas en el fichero
str(hatco)
```

```
## List of 15
##  $ DELSPEED: num [1:100] 4.1 1.8 2.7 4.6 2.4 2.8 3.7 3.4 2.4 2.4 ...
##  $ PRICELEV: num [1:100] 0.6 3 1 2.4 1.6 1.4 1.5 0.4 1.5 1.5 ...
##  $ PRICEFLE: num [1:100] 6.9 6.3 7.1 9.5 8.8 8.1 8.6 8.3 6.7 6.6 ...
##  $ MANUFIMA: num [1:100] 4.7 6.6 5.9 6.6 4.8 3.8 5.7 2.5 4.8 4.8 ...
##  $ SERVICE : num [1:100] 2.4 2.5 1.8 3.5 2 2.1 2.7 1.2 1.9 1.9 ...
##  $ SALESFOR: num [1:100] 2.3 4 2.3 4.5 2.8 1.4 3.7 1.7 2.5 2.5 ...
##  $ PRODUCTQ: num [1:100] 5.2 8.4 7.8 7.6 5.8 6.6 6.7 5.2 7.2 7.2 ...
##  $ TAMEMP  : Factor w/ 2 levels "Small","Large": 1 2 2 1 1 2 1 1 2 2 ...
##  $ USAGELEV: num [1:100] 32 43 32 46 32 39 38 35 36 36 ...
##  $ SATISFLE: num [1:100] 4.2 4.3 3.9 5.8 4.3 4.4 5 3.3 3.7 3.7 ...
##  $ ESPCOMPR: Factor w/ 2 levels "Specification Buying",..: 2 1 1 2 2 1 2 2 1 1 ...
##  $ ESTRCOMP: Factor w/ 2 levels "Decentralized",..: 1 2 2 1 1 2 1 1 2 2 ...
##  $ INDUSTRI: Factor w/ 2 levels "Other Industries",..: 2 1 2 2 1 1 2 1 1 1 ...
##  $ SITCOMP : Factor w/ 3 levels "New Task","Modified Rebuy",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ RANDZ   : num [1:100] 0 0 1 0 0 0 0 1 0 1 ...
##  - attr(*, "label.table")=List of 15
##   ..$ DELSPEED: NULL
##   ..$ PRICELEV: NULL
##   ..$ PRICEFLE: NULL
##   ..$ MANUFIMA: NULL
##   ..$ SERVICE : NULL
##   ..$ SALESFOR: NULL
##   ..$ PRODUCTQ: NULL
##   ..$ TAMEMP  : Named num [1:2] 1 0
##   .. ..- attr(*, "names")= chr [1:2] "Large" "Small"
##   ..$ USAGELEV: NULL
##   ..$ SATISFLE: NULL
##   ..$ ESPCOMPR: Named num [1:2] 1 0
##   .. ..- attr(*, "names")= chr [1:2] "Total Value Analysis" "Specification Buying"
##   ..$ ESTRCOMP: Named num [1:2] 1 0
##   .. ..- attr(*, "names")= chr [1:2] "Centralized" "Decentralized"
##   ..$ INDUSTRI: Named num [1:2] 1 0
##   .. ..- attr(*, "names")= chr [1:2] "Industry A Classification" "Other Industries"
##   ..$ SITCOMP : Named num [1:3] 3 2 1
##   .. ..- attr(*, "names")= chr [1:3] "Straight Rebuy" "Modified Rebuy" "New Task"
##   ..$ RANDZ   : NULL
##  - attr(*, "variable.labels")= Named chr [1:15] "Delivery Speed" "Price Level" "Price Flexibility" "
##   ..- attr(*, "names")= chr [1:15] "DELSPEED" "PRICELEV" "PRICEFLE" "MANUFIMA" ...
```

En estas líneas de código mostramos también cómo leer los datos en el formato `csv` de intercambio de hojas electrónicas, donde las columnas están separadas por `;` y el símbolo de los decimales es la coma, `,` en lugar del punto decimal anglosajón.

Selección de las bases de segmentación

Ahora deberás seleccionar las bases de segmentación que utilizarás con los programas de aglomeración jerárquica y partición. En el ejemplo se han seleccionado las primeras 7. Puedes seleccionar éstas o cualesquiera otras, pero deberás, en cualquier caso, argumentar por qué razón escoges esas variables como bases de segmentación.

```
#require(dplyr)
bases<-data.frame(hatco[1:7])
names(bases)
```

```
## [1] "DELSPEED" "PRICELEV" "PRICEFLE" "MANUFIMA" "SERVICE"
## [6] "SALESFOR" "PRODUCTQ"
```

Ahora deberás tomar otra decisión sobre las bases de segmentación. Se trata de decidir si es necesario hacer alguna transformación en ellas antes de utilizar los algoritmos de clasificación. Si no estás seguro o segura de si es necesario puedes hacer el proceso con las variables transformadas y las originales y observar el resultado de la clasificación. Si coincide, bien, si no entonces deberás tomar una decisión al respecto.

Comprobar que las bases están medidas en la misma escala. Si no lo estuvieran, entonces normalizar las bases de segmentación utilizando la función scale() o bases.norm<-scale(bases)

Si quieres conocer las correlaciones, sólo tienes que utilizar el siguiente comando:

```
cor(bases)
```

```
##             DELSPEED   PRICELEV    PRICEFLE   MANUFIMA
## DELSPEED  1.00000000 -0.3492251  0.50929519  0.0504142
## PRICELEV -0.34922515  1.0000000 -0.48721259  0.2721868
## PRICEFLE  0.50929519 -0.4872126  1.00000000 -0.1161041
## MANUFIMA  0.05041420  0.2721868 -0.11610408  1.0000000
## SERVICE   0.61190069  0.5129808  0.06661728  0.2986774
## SALESFOR  0.07711522  0.1862433 -0.03431610  0.7882245
## PRODUCTQ -0.48263094  0.4697458 -0.44811201  0.1999811
##              SERVICE    SALESFOR   PRODUCTQ
## DELSPEED  0.61190069  0.07711522 -0.4826309
## PRICELEV  0.51298082  0.18624325  0.4697458
## PRICEFLE  0.06661728 -0.03431610 -0.4481120
## MANUFIMA  0.29867737  0.78822454  0.1999811
## SERVICE   1.00000000  0.24080818 -0.0551613
## SALESFOR  0.24080818  1.00000000  0.1772939
## PRODUCTQ -0.05516130  0.17729392  1.0000000
```

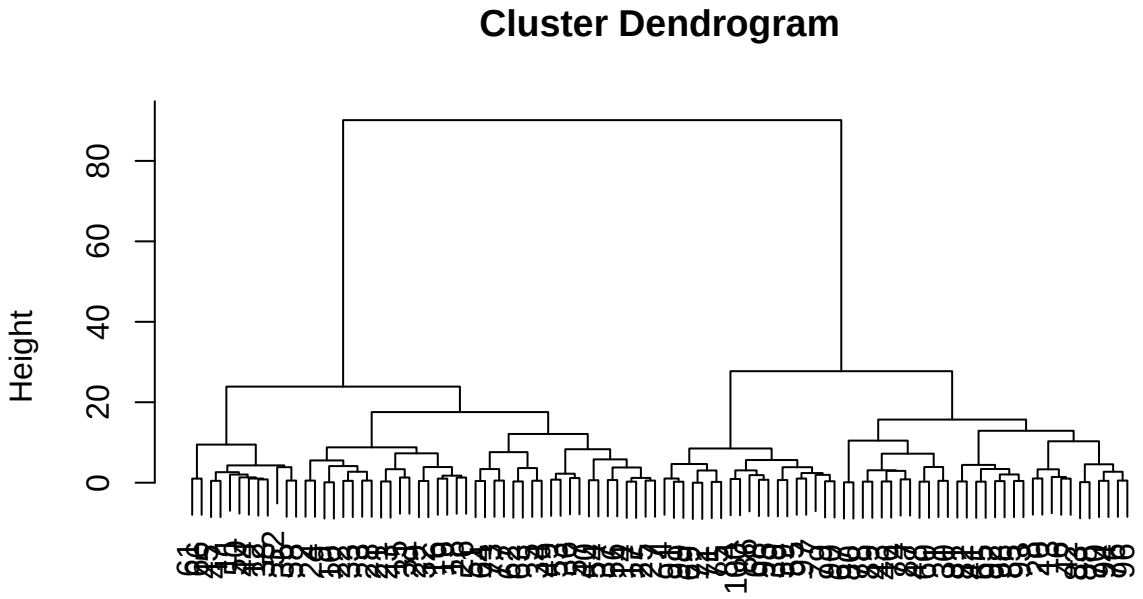
Valoración de la heterogeneidad en la muestra: selección del número de segmentos

Agrupamos a los clientes y mostramos el resultado de la agrupación

```
bases.hclust<-hclust(dist(bases, method="euclidean"), method="ward")
```

```
## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
```

```
#Mostramos el resultado de la agrupación
plot(bases.hclust)
```



Ahora calculamos los centros de los grupos formados durante el proceso de agrupación jerárquica y los mostramos.

```
centros.bases<-tapply(as.matrix(bases), list(rep(cutree(bases.hclust, 2), ncol(as.matrix(bases))),col(as
#Visualizamos el resultado
centros.bases
```

```
##      1     2     3     4     5    6     7
## 1 4.46 1.576 8.900 4.926 2.992 2.51 5.904
## 2 2.57 3.152 6.888 5.570 2.840 2.82 8.038
```

Partición de la muestra en el número de segmentos seleccionados

Dividimos la muestra con kmeans

```
bases.kmeans2<-kmeans(bases, centros.bases)
```

Para caracterizar a los segmentos utilizamos las medias de las variables originales en los segmentos formados.

```
names(bases.kmeans2)
```

```
## [1] "cluster"      "centers"      "totss"
## [4] "withinss"     "tot.withinss" "betweenss"
## [7] "size"         "iter"         "ifault"
```

El objeto centers contiene la información que buscamos si hemos segmentado con las variables originales

```
bases.kmeans2$centers
```

```
##   DELSPEED PRICELEV PRICEFLE MANUFIMA  SERVICE SALESFOR
## 1 4.382692 1.580769 8.900000 4.925000 2.957692 2.525000
## 2 2.575000 3.212500 6.804167 5.597917 2.870833 2.816667
##   PRODUCTQ
## 1 5.903846
## 2 8.127083
```

Descripción de los segmentos formados

Identificamos a los componentes de los grupos. Para ello utilizamos las variables descriptoras de los segmentos

```
table(hatco$TAMEMP, bases.kmeans2$cluster)
```

```
##
##          1  2
##   Small 50 10
##   Large  2 38
```

Como USAGELEV es una variable metrica podemos calcular las medias con la función t.test() o con la función lm()

```
t.test(hatco$USAGELEV ~ bases.kmeans2$cluster)
```

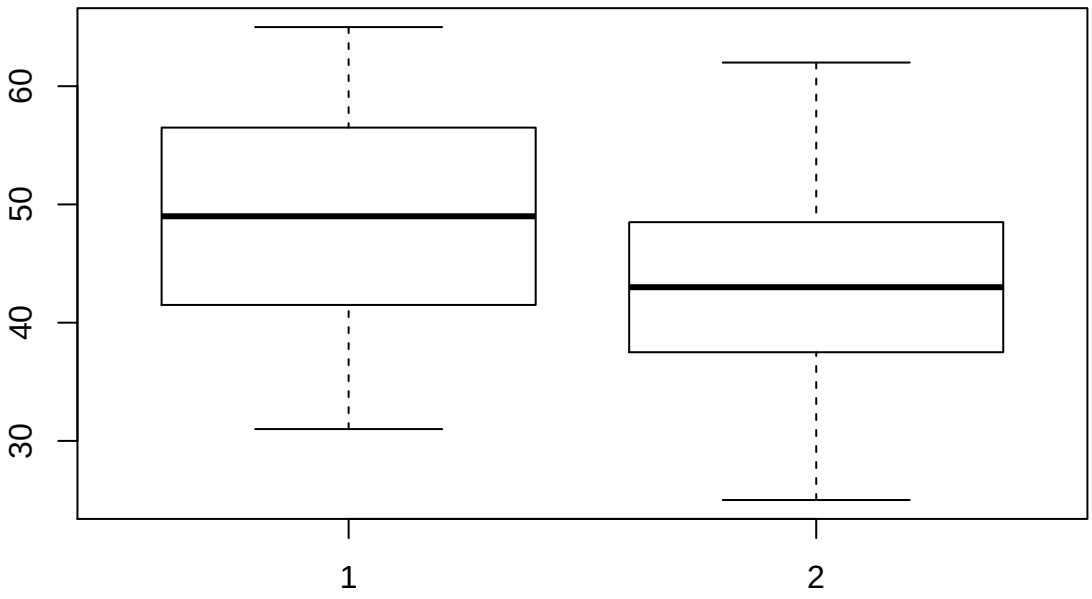
```
##
##  Welch Two Sample t-test
##
## data:  hatco$USAGELEV by bases.kmeans2$cluster
## t = 3.8704, df = 97.393, p-value = 0.0001966
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  3.158448 9.806295
## sample estimates:
## mean in group 1 mean in group 2
##        49.21154        42.72917
```

```
summary(lm(hatco$USAGELEV ~ bases.kmeans2$cluster))
```

```
##
## Call:
## lm(formula = hatco$USAGELEV ~ bases.kmeans2$cluster)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.2115  -5.9792   0.2708   6.2708  19.2708
##
## Coefficients:
##                         Estimate Std. Error t value
## (Intercept)               55.694      2.633  21.152
## bases.kmeans2$cluster     -6.482      1.686  -3.846
##                         Pr(>|t|)
## (Intercept)              < 2e-16 ***
## bases.kmeans2$cluster 0.000214 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.421 on 98 degrees of freedom
## Multiple R-squared:  0.1311, Adjusted R-squared:  0.1223
## F-statistic: 14.79 on 1 and 98 DF,  p-value: 0.0002139
```

Y ver las medias en un gráfico como el siguiente

```
boxplot(hatco$USAGELEV ~ bases.kmeans2$cluster)
```



Para las variables categóricas utilizamos la función table()

```
table(hatco$ESPCOMPR, bases.kmeans2$cluster)
```

```
##
##                          1  2
##   Specification Buying   2 38
##   Total Value Analysis  50 10
```

```
table(hatco$ESTRCOMP, bases.kmeans2$cluster)
```

```
##
##                  1  2
##   Decentralized 50  0
##   Centralized    2 48
```

```
table(hatco$INDUSTRI, bases.kmeans2$cluster)
```

```
##
##                              1  2
##   Other Industries          28 22
##   Industry A Classification 24 26
```

```
table(hatco$SITCOMP, bases.kmeans2$cluster)
```

```
##
##                   1  2
##   New Task       10 24
##   Modified Rebuy 10 22
##   Straight Rebuy 32  2
```

También puedes contruir una tabla con la información de las variables descriptoras de los segmentos

```
descriptores<-data.frame(hatco[8:14])
names(descriptores)
```

```
## [1] "TAMEMP"   "USAGELEV" "SATISFLE" "ESPCOMPR" "ESTRCOMP"
## [6] "INDUSTRI" "SITCOMP"
```

```
#Mostramos el valor medio de las bases de segmentación en los grupos
options(digits=3)
t(aggregate(descriptores, list(Segmento = bases.kmeans2$cluster), mean))
```

```
## Warning in mean.default(X[[1L]], ...): argument is not
## numeric or logical: returning NA
```

```
## Warning in mean.default(X[[2L]], ...): argument is not
## numeric or logical: returning NA
```

```
## Warning in mean.default(X[[1L]], ...): argument is not
## numeric or logical: returning NA
```

```
## Warning in mean.default(X[[2L]], ...): argument is not
## numeric or logical: returning NA
```

```
## Warning in mean.default(X[[1L]], ...): argument is not
## numeric or logical: returning NA
```

```
## Warning in mean.default(X[[2L]], ...): argument is not
## numeric or logical: returning NA

## Warning in mean.default(X[[1L]], ...): argument is not
## numeric or logical: returning NA

## Warning in mean.default(X[[2L]], ...): argument is not
## numeric or logical: returning NA

## Warning in mean.default(X[[1L]], ...): argument is not
## numeric or logical: returning NA

## Warning in mean.default(X[[2L]], ...): argument is not
## numeric or logical: returning NA
```

```
##           [,1]  [,2]
## Segmento  1.00  2.00
## TAMEMP      NA    NA
## USAGELEV 49.21 42.73
## SATISFLE  5.13  4.38
## ESPCOMPR    NA    NA
## ESTRCOMP    NA    NA
## INDUSTRI    NA    NA
## SITCOMP     NA    NA
```

Transformación de las bases de segmentación

Ahora comprobariamos si la segmentación obtenida con las bases originales varía cuando las sustituimos por sus componentes principales

Comprobar que la correlación entre las bases de segmentación no suponga un problema para la segmentacion. Visualizamos las correlaciones entre las bases de segmentación

```
cor(bases)
```

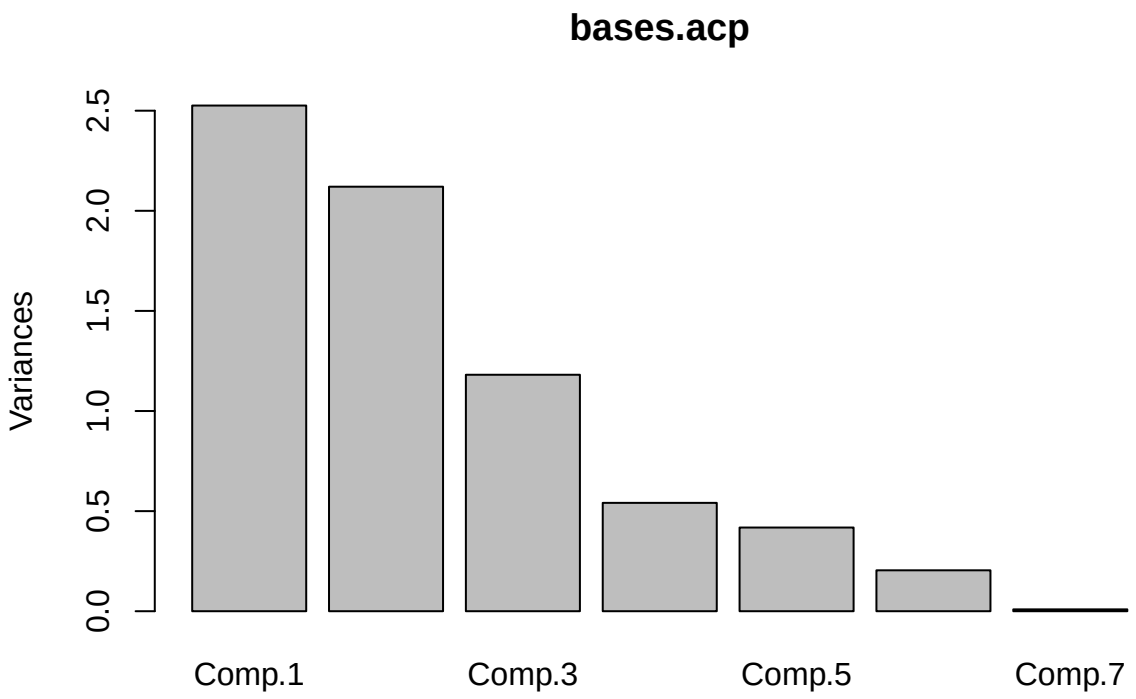
```
##          DELSPEED PRICELEV PRICEFLE MANUFIMA SERVICE
## DELSPEED   1.0000   -0.349   0.5093   0.0504  0.6119
## PRICELEV  -0.3492    1.000  -0.4872   0.2722  0.5130
## PRICEFLE   0.5093   -0.487   1.0000  -0.1161  0.0666
## MANUFIMA   0.0504    0.272  -0.1161   1.0000  0.2987
## SERVICE    0.6119    0.513   0.0666   0.2987  1.0000
## SALESFOR   0.0771    0.186  -0.0343   0.7882  0.2408
## PRODUCTQ  -0.4826    0.470  -0.4481   0.2000 -0.0552
##          SALESFOR PRODUCTQ
## DELSPEED   0.0771  -0.4826
## PRICELEV   0.1862   0.4697
## PRICEFLE  -0.0343  -0.4481
## MANUFIMA   0.7882   0.2000
## SERVICE    0.2408  -0.0552
## SALESFOR   1.0000   0.1773
## PRODUCTQ   0.1773   1.0000
```

Si la correlación es elevada, transformamos las bases en unas nuevas variables llamadas componentes principales. Ahora vamos a calcular los componentes principales para comprobar si el resutlado cambia

```
bases.acp<-princomp(bases, cor=T)
```

Podemos visualizar la varianza explicada por cada componente principal. Al utilizar las correlación (acotadas entre -1 y 1) la variacion explicada por los componentes principales es igual al número de variables originales y la varianza explicada por cada componentes estará entre 0 y 7

```
plot(bases.acp)
```




```
summary(bases.acp)
```

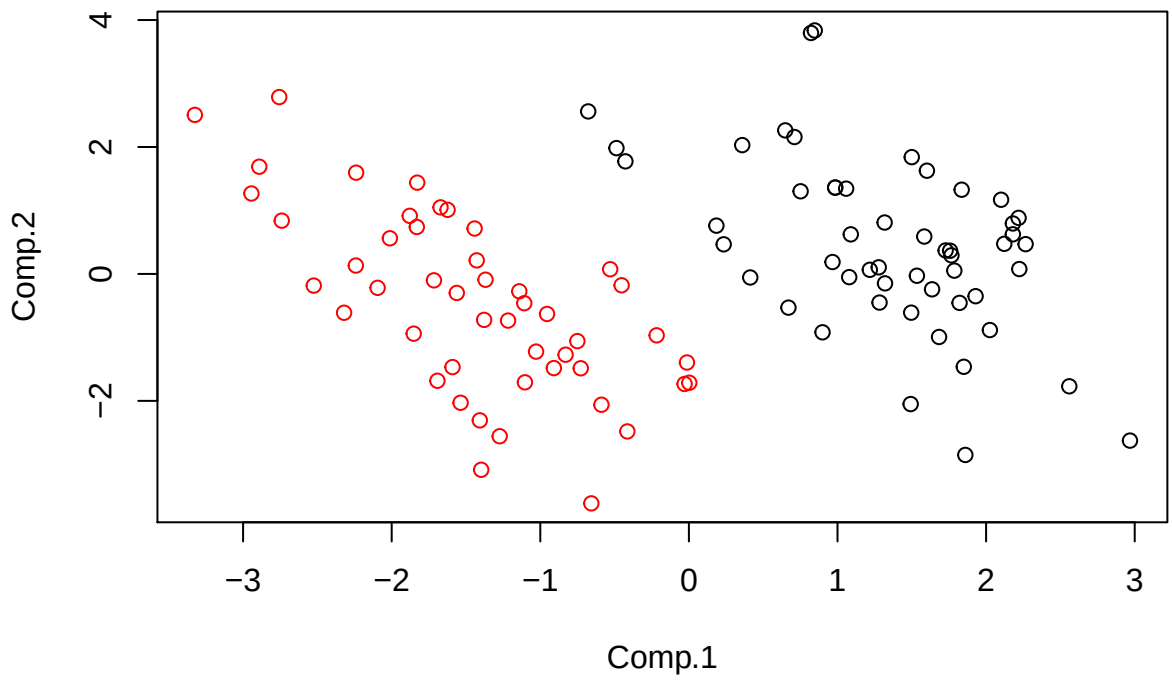
```
## Importance of components:
##                          Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
## Standard deviation      1.589  1.456  1.087 0.7356 0.6466
## Proportion of Variance  0.361  0.303  0.169 0.0773 0.0597
## Cumulative Proportion   0.361  0.664  0.832 0.9098 0.9695
##                          Comp.6  Comp.7
## Standard deviation     0.4521 0.09571
## Proportion of Variance 0.0292 0.00131
## Cumulative Proportion  0.9987 1.00000
```

Vemos que con cuatro cuatro componentes explicamos el 90% de la variación. Ahora para segmentar utilizamos la puntuación de los clientes en los nuevos componentes principales Esa puntuación está recogida en el objeto scores de la lista bases.acp La asignamos al objeto bases.puntos

```
bases.puntos<-bases.acp$scores
```

Si queremos visualizar el resultado de la clasificacion que hemos realizado antes, podemos utilizar los componentes de esta forma:

```
plot(bases.puntos[,1:2], col=bases.kmeans2$cluster)
```

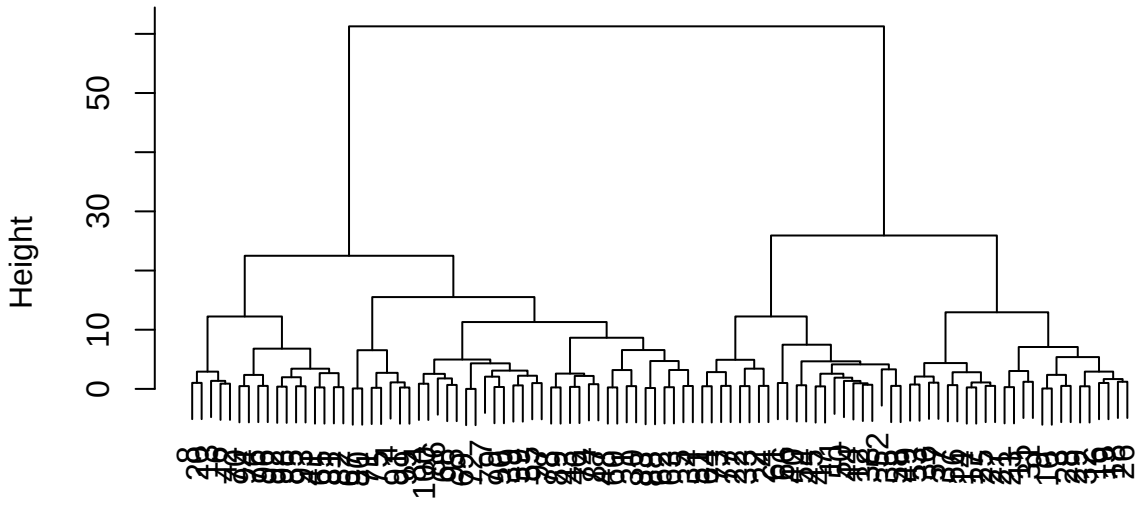


#Segmentación con las bases transformadas en sus componentes principales Ahora volvemos a realizar el proceso de segmentación, pero con los componentes principales. Primero la agrupación jerárquica

```
bases.puntos.hclust<-hclust(dist(bases.puntos), method="ward")
```

```
## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
```

```
#visualizamos la heterogeneidad y la comparamos con el la mostrada con los datos originales
plot(bases.puntos.hclust)
```




```
#Calculamos los centros de los grupos
centros.bases.puntos<-tapply(bases.puntos, list(rep(cutree(bases.puntos.hclust, 2), ncol(bases.puntos))
#isualizamos los centros
centros.bases.puntos
```

```
##       1      2       3       4       5       6       7
## 1  1.26  0.429  0.0696  0.0373 -0.0407  0.0100  0.00271
## 2 -1.48 -0.503 -0.0816 -0.0438  0.0478 -0.0118 -0.00318
```

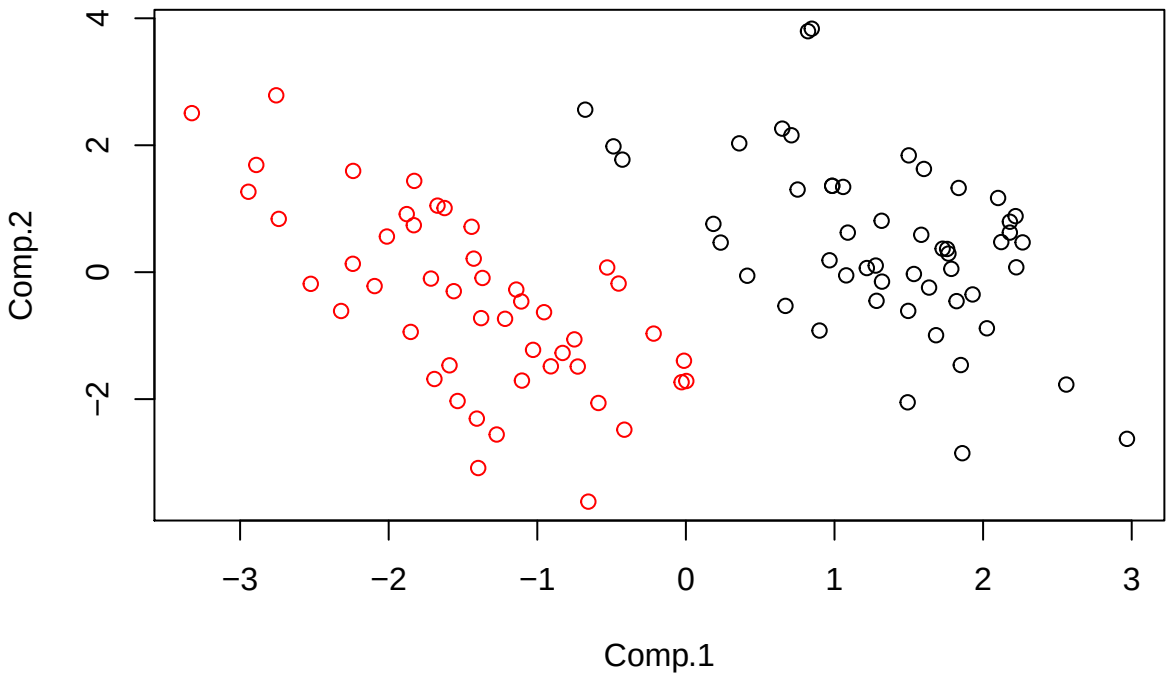
De nuevo partimos la muestra con kmeans

```
bases.puntos.kmeans2<-kmeans(bases.puntos, centros.bases.puntos)
bases.puntos.kmeans2
```

```
## K-means clustering with 2 clusters of sizes 52, 48
##
## Cluster means:
##   Comp.1 Comp.2  Comp.3  Comp.4   Comp.5  Comp.6   Comp.7
## 1   1.33  0.447  0.0563  0.0376 -0.00548  0.0248  0.00226
## 2  -1.44 -0.484 -0.0610 -0.0408  0.00593 -0.0269 -0.00245
##
## Clustering vector:
##   [1] 1 2 2 1 1 1 1 1 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2
##  [27] 2 2 2 1 2 2 2 1 2 2 2 2 2 1 2 1 1 1 2 2 1 2 1 2 2 2 2
##  [53] 2 2 1 2 2 2 1 1 2 2 2 2 1 2 1 1 1 1 1 1 2 1 1 1 2 1
##  [79] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##
```

```
## Within cluster sum of squares by cluster:
## [1] 251 234
##  (between_SS / total_SS =  30.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"
## [4] "withinss"     "tot.withinss" "betweenss"
## [7] "size"         "iter"         "ifault"
```

```
plot(bases.puntos[,1:2], col=bases.puntos.kmeans2$cluster)
```



Y

Comparamos el resultado

```
table(bases.kmeans2$cluster, bases.puntos.kmeans2$cluster)
```

```
##
##      1  2
##   1 52  0
##   2  0 48
```

Ahora los centros de los segmentos que nos proporciona el objeto bases.puntos.kmeans2 corresponden a los componentes principales utilizados en elanálisis. Si queremos obtener los valores medios de las bases en los segmentos, tenemos que calcularlos de esta manera:

```
options(digits=3)
#Mostramos el valor medio de las bases de segmentación en los grupos
t(aggregate(bases, list(Segmento = bases.puntos.kmeans2$cluster), mean))
```

```
##          [,1] [,2]
## Segmento 1.00 2.00
```

```
## DELSPEED 4.38 2.58
## PRICELEV 1.58 3.21
## PRICEFLE 8.90 6.80
## MANUFIMA 4.92 5.60
## SERVICE  2.96 2.87
## SALESFOR 2.52 2.82
## PRODUCTQ 5.90 8.13
```

Decisiones estratégicas