

tarea3

Karina De Sousa

9 de abril de 2016

Objectivo:

Escoger, en base a los conocimientos adquiridos en clase, el mejor algoritmo de clustering según su criterio para distintos datasets dados por el grupo docente.

Solución:

Cargamos los paquetes necesarios,

```
## [1] "scatterplot3d" "gmodels"      "stats"       "graphics"  
## [5] "grDevices"     "utils"        "datasets"    "methods"  
## [9] "base"
```

a.csv

1. Cargamos el dataset *a.csv*, asumiendo que se hizo `setwd(directorio)`:

```
> aFile = read.csv(file = "./a.csv", header = FALSE, sep = ",")  
> head(aFile)
```

```
##          V1      V2 V3  
## 1 -15.14267 -18.121187 1  
## 2 13.05504   9.445506 0  
## 3 -13.61853  -8.141030 1  
## 4 -14.36733 -11.119844 1  
## 5 -11.63675 -13.343746 1  
## 6 -10.64171 -12.324683 1
```

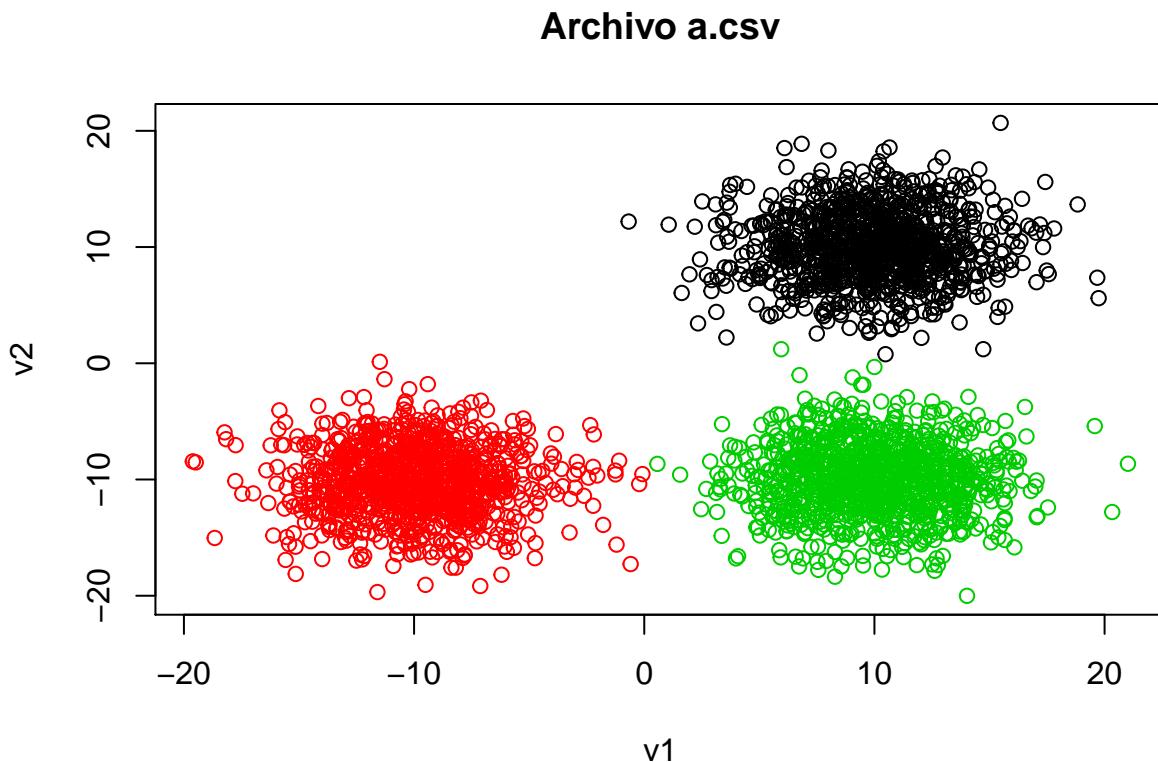
2. Separamos la última columna del dataset. La cual, hace referencia a la clase de cada instancia,

```
> aClass = aFile$V3  
> aFile = aFile[c("V1", "V2")]  
> head(aFile)
```

```
##          V1      V2  
## 1 -15.14267 -18.121187  
## 2 13.05504   9.445506  
## 3 -13.61853  -8.141030  
## 4 -14.36733 -11.119844  
## 5 -11.63675 -13.343746  
## 6 -10.64171 -12.324683
```

3. Ahora, hacemos un *plot* para verificar como se comportan los datos de *a.csv*,

```
> # Sumamos uno a aClass porque la clase cero es el color blanco  
> plot(aFile$V1, aFile$V2, col = aClass + 1, xlab = "v1", ylab = "v2", main = "Archivo a.csv")
```

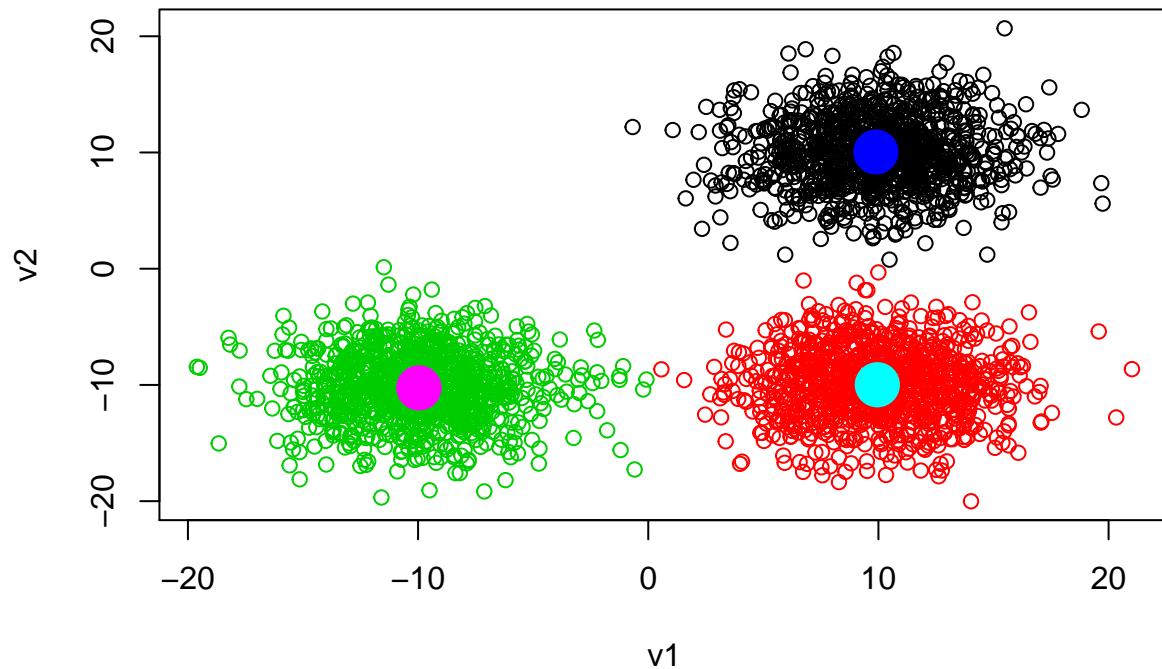


Vemos que los datos se separan en 3 clusters circulares bien definidos y basandonos en la teoría, k medias es la mejor forma de predecir las clases de este tipo de datasets.

4. Usamos la función kmeans y verificamos que tan bien predice las clases del dataset con **k=3**, porque este fue el número de clusters hallados en el análisis exploratorio,

```
> aFile.k.medias = kmeans(x = aFile, centers = 3)  
>  
> # Graficamos los clusters obtenidos en kmeans  
> plot(aFile$V1, aFile$V2, col = aFile.k.medias$cluster, xlab = "v1", ylab = "v2",  
+       main = "Archivo a.csv")  
>  
> # Graficamos los centroides asociados a cada cluster Usamos un color  
# distinto para los centroides para verlos mejor  
>  
> # Para a_big.csv  
> aCentroids = aFile.k.medias$centers  
>  
> points(x = aFile.k.medias$centers[, c("V1", "V2")], col = 4:6, pch = 19, cex = 3)
```

Archivo a.csv



5. Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = aFile.k.medias$cluster, y = aClass, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
## |-----|  
## |           N |  
## |           N / Row Total |  
## |           N / Col Total |  
## |           N / Table Total |  
## |-----|  
##  
##  
## Total Observations in Table:  3000  
##  
##  
##          | aClass  
## aFile.k.medias$cluster |     0 |      1 |      2 | Row Total |  
## -----|-----|-----|-----|-----|  
##     1 |    1000 |      0 |      1 |    1001 |  
##           | 0.999 | 0.000 | 0.001 | 0.334 |  
##           | 1.000 | 0.000 | 0.001 |       |  
##           | 0.333 | 0.000 | 0.000 |       |
```

```

## -----|-----|-----|-----|-----|
##      2 |     0 |     0 |    999 |    999 |
##          | 0.000 | 0.000 | 1.000 | 0.333 |
##          | 0.000 | 0.000 | 0.999 |   |
##          | 0.000 | 0.000 | 0.333 |   |
## -----|-----|-----|-----|-----|
##      3 |     0 |   1000 |     0 |   1000 |
##          | 0.000 | 1.000 | 0.000 | 0.333 |
##          | 0.000 | 1.000 | 0.000 |   |
##          | 0.000 | 0.333 | 0.000 |   |
## -----|-----|-----|-----|-----|
## Column Total |   1000 |   1000 |   1000 | 3000 |
##          | 0.333 | 0.333 | 0.333 |   |
## -----|-----|-----|-----|-----|
##
```

Vemos que las etiquetas de las clases asignadas por *kmeans* son distintas a las de *aClass*. Podemos modificar esto para que la matriz de confusión sea más fácil de leer,

```

> aClusters = aFile.k.medias$cluster
> aClusters[aClusters == 1] = 4
> aClusters[aClusters == 2] = 5
> aClusters[aClusters == 3] = 6
>
> aClusters[aClusters == 4] = 2
> aClusters[aClusters == 5] = 1
> aClusters[aClusters == 6] = 0
>
> CrossTable(x = aClusters, y = aClass, prop.chisq = FALSE)

```

```

## 
## 
##   Cell Contents
## |-----|
## |           N |
## |           N / Row Total |
## |           N / Col Total |
## |           N / Table Total |
## |-----|
## 
## 
## Total Observations in Table: 3000
## 
## 
##           | aClass
##   aClusters |     0 |     1 |     2 | Row Total |
## -----|-----|-----|-----|-----|
##      0 |     0 | 1000 |     0 |   1000 |
##          | 0.000 | 1.000 | 0.000 | 0.333 |
##          | 0.000 | 1.000 | 0.000 |   |
##          | 0.000 | 0.333 | 0.000 |   |
## -----|-----|-----|-----|-----|

```

```

##      1 |      0 |      0 |    999 |    999 |
##      | 0.000 | 0.000 | 1.000 | 0.333 |
##      | 0.000 | 0.000 | 0.999 |
##      | 0.000 | 0.000 | 0.333 |
## -----
##      2 | 1000 |      0 |      1 | 1001 |
##      | 0.999 | 0.000 | 0.001 | 0.334 |
##      | 1.000 | 0.000 | 0.001 |
##      | 0.333 | 0.000 | 0.000 |
## -----
## Column Total | 1000 | 1000 | 1000 | 3000 |
##      | 0.333 | 0.333 | 0.333 |
## -----
##
```

Vemos que el algoritmo solo se equivoca con un punto que originalmente es de la clase 2 y es clasificado como parte de la clase 0. Haciendo de k medias el mejor algoritmos para el dataset **a.csv**.

moon.csv

1. Cargamos el dataset **moon.csv**, asumiendo que se hizo `setwd(directorio)`:

```

> moonFile = read.csv(file = "./moon.csv", header = FALSE, sep = ",")
> head(moonFile)

```

```

##          V1         V2 V3
## 1  0.438270  0.900774  0
## 2 -0.767655  0.672376  0
## 3  0.487233  0.827758  0
## 4  0.904183 -0.446244  1
## 5  0.190063 -0.142372  1
## 6  0.113667  0.097442  1

```

2. Separamos la última columna del dataset. La cual, hace referencia a la clase de cada instancia,

```

> moonClass = moonFile$V3
> moonFile = moonFile[c("V1", "V2")]
> head(moonFile)

```

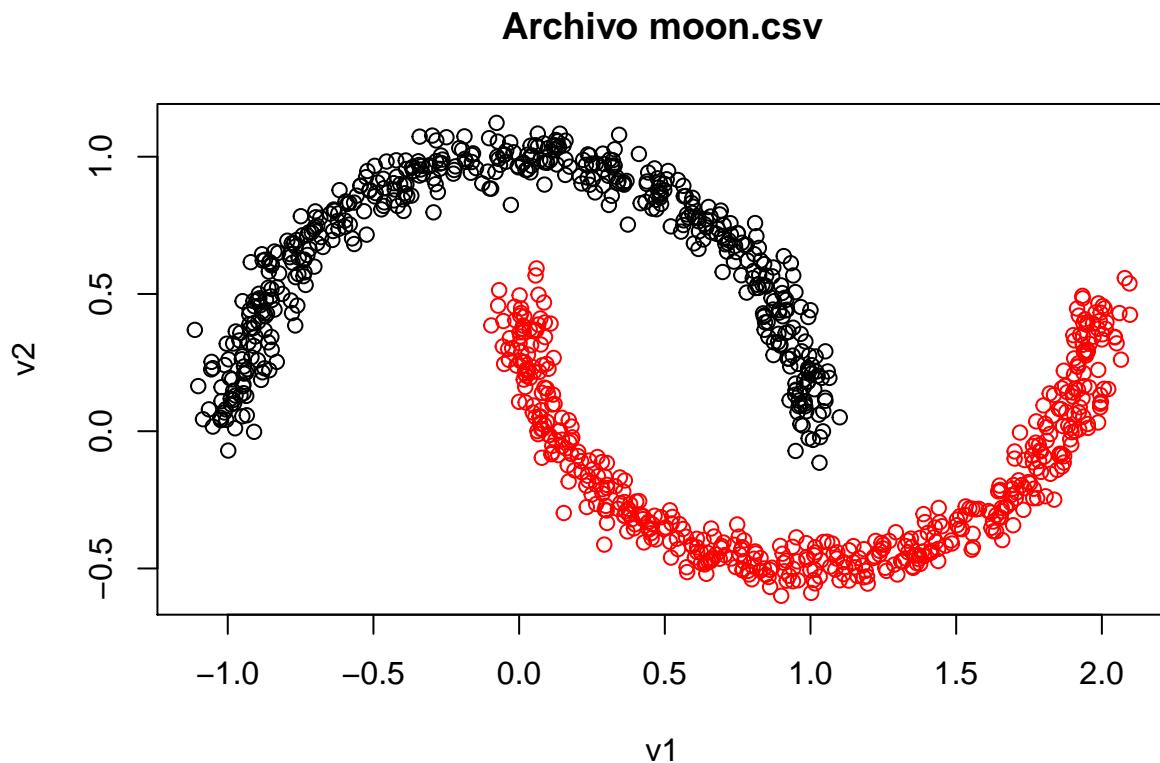
```

##          V1         V2
## 1  0.438270  0.900774
## 2 -0.767655  0.672376
## 3  0.487233  0.827758
## 4  0.904183 -0.446244
## 5  0.190063 -0.142372
## 6  0.113667  0.097442

```

3. Ahora, hacemos un *plot* para verificar como se comportan los datos de **a.csv**,

```
> # Sumamos uno a moonClass porque la clase cero es el color blanco
> plot(moonFile$V1, moonFile$V2, col = moonClass + 1, xlab = "v1", ylab = "v2",
+       main = "Archivo moon.csv")
```



Vemos que los datos se separan en 2 clusters con forma de media luna, por lo que un algoritmos jerárquico será la mejor forma de predecir las clases de este dataset.

4. Para usar la función *hclust* debemos seleccionar el metodo de aglomeracion, en este caso usaremos la norma 2. Para esto, calculamos la misma usando la función *dist*.

Para *hclust* podemos usar *complete link* o *single link*,

- Con *sinlge link* la similaridad entre dos clusters es la mayor similaridad entre dos de sus miembros. Este método se ajusta a la forma de los clusters.
- Con *complete link* la similaridad entre dos clusters es la menor similaridad entre dos de sus miembros. Este método funciona muy bien para clusters verticales.

Basados en la teoría y en la forma de los clusters de **moon.csv**, usaremos *single link*.

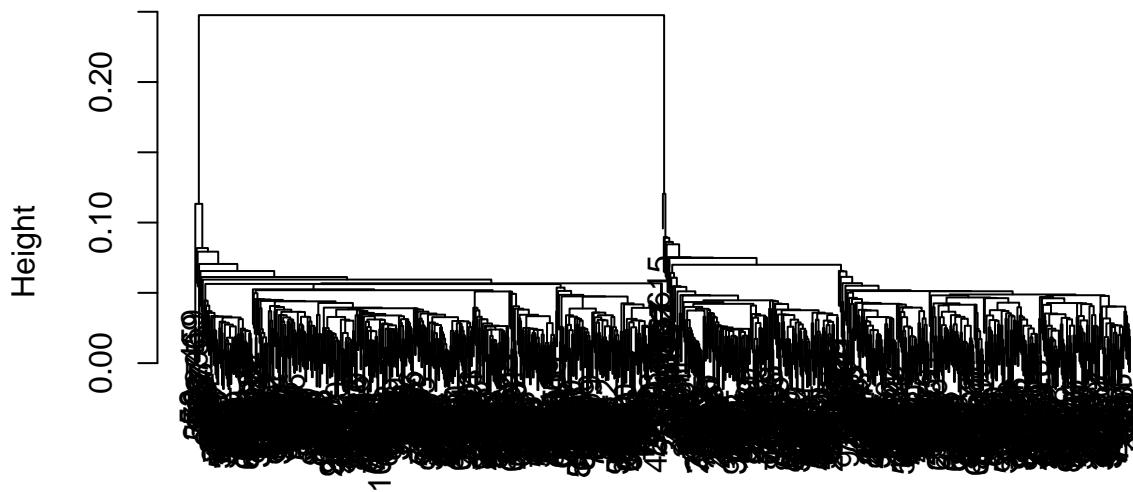
```
> # Dataframe a Matrix
> moonFileMatrix = as.matrix(moonFile)
>
> # Matriz de distancia con norma 2 method = 'euclidean', 'maximum',
> # 'manhattan', 'canberra', 'binary' or 'minkowski'
```

```

> distancia = dist(moonFileMatrix)
>
> # Método por defecto es complete link
> #'single', 'complete'
> metodo = "single"
>
> cluster = hclust(distancia, method = metodo)
>
> plot(cluster, main = "Cluster jerarquico de moon.csv")

```

Cluster jerarquico de moon.csv



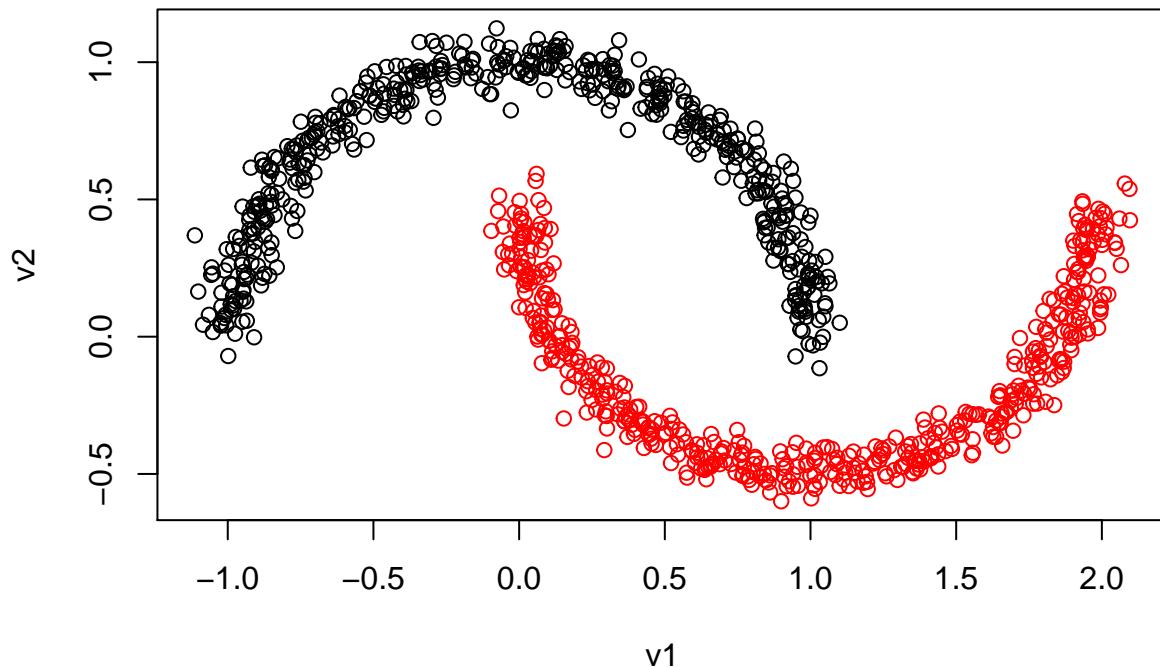
5. A partir del dendograma obtenido en el punto anterior, determinamos las clases que el método asignó a cada instancia usando *cutree*.

```

> # En el análisis exploratorio observamos 2 clusters, por lo que este será el
> # número de clases
> nclases = 2
> # Cortamos
> corte = cutree(cluster, k = nclases)
>
> # Graficamos el dataset con el color de las clases obtenidas con hclust
> plot(moonFile$V1, moonFile$V2, col = corte, xlab = "v1", ylab = "v2", main = "Archivo moon.csv hclust")

```

Archivo moon.csv hclust single



6. Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = corte, y = moonClass + 1, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
## |-----|  
## |           N |  
## |           N / Row Total |  
## |           N / Col Total |  
## |           N / Table Total |  
## |-----|  
##  
##  
## Total Observations in Table:  1000  
##  
##  
##          | moonClass + 1  
##  corte |      1 |      2 | Row Total |  
## -----|-----|-----|-----|  
##    1 |    500 |     0 |    500 |  
##    | 1.000 | 0.000 | 0.500 |  
##    | 1.000 | 0.000 |     |  
##    | 0.500 | 0.000 |     |
```

```

## -----|-----|-----|-----|
##      2 |     0 |    500 |    500 |
##      | 0.000 | 1.000 | 0.500 |
##      | 0.000 | 1.000 |      |
##      | 0.000 | 0.500 |      |
## -----|-----|-----|-----|
## Column Total |    500 |    500 |   1000 |
##      | 0.500 | 0.500 |      |
## -----|-----|-----|-----|
##
```

Vemos que todos los datos fueron clasificados de forma correcta.

h.csv

1. Cargamos el dataset *h.csv*, asumiendo que se hizo `setwd(directorio)`:

```
> hFile = read.csv(file = "./h.csv", header = FALSE, sep = ",")  
> head(hFile)
```

```

##      V1        V2        V3        V4
## 1 -8.867581 12.523203 -4.358825 9.884834
## 2  5.193664  0.207447 -10.064941 11.452890
## 3 -6.053527  9.987890 -8.423283 10.393300
## 4 -8.964402 14.910192 -3.891926 9.847792
## 5 -6.705757  0.916498  5.818809 8.705241
## 6 -2.016423 18.494963 -10.528826 10.799798

```

```
> summary(hFile$V4)
```

```

##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 4.718 7.046  9.249  9.386 11.660 14.140

```

2. La última columna de *h.csv* contiene las clases asociadas a las instancias del dataset, pero estos valores se encuentran en el intervalo (4,15) por lo que debemos transformarlas y para esto implementamos la función `definir_clase`

```
> definir_clase = function(numero) {  
+   if (numero <= 6)  
+     return(1) else if (numero > 6 && numero <= 8)  
+     return(2) else if (numero > 8 && numero <= 10)  
+     return(3) else if (numero > 10 && numero <= 12)  
+     return(4) else return(5)  
+ }
```

Ahora, usamos esta función para asignar una clase a cada instancia del dataset,

```
> for (i in 1:length(hFile$V4)) {  
+   hFile$V4[i] = definir_clase(hFile$V4[i])  
+ }  
>  
> # hFile$V4
```

3. Separamos la última columna del dataset. La cual, hace referencia a la clase de cada instancia,

```
> hClass = hFile$V4  
> hFile = hFile[c("V1", "V2", "V3")]  
> head(hFile)
```

```
##           V1          V2          V3  
## 1 -8.867581 12.523203 -4.358825  
## 2  5.193664  0.207447 -10.064941  
## 3 -6.053527  9.987890 -8.423283  
## 4 -8.964402 14.910192 -3.891926  
## 5 -6.705757  0.916498  5.818809  
## 6 -2.016423 18.494963 -10.528826
```

Vemos cuantas clases contiene el dataset

```
> table(hClass)
```

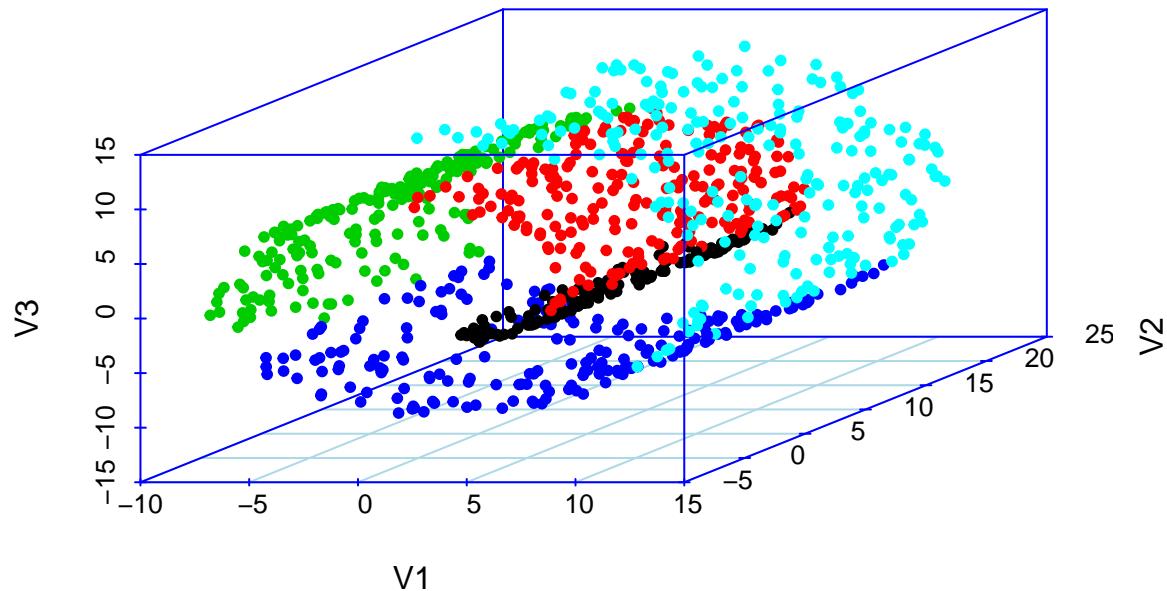
```
## hClass  
##   1   2   3   4   5  
## 140 219 214 201 226
```

El dataset posee 5 clases.

4. Ahora graficamos el dataset, para verificar como se comportan los datos

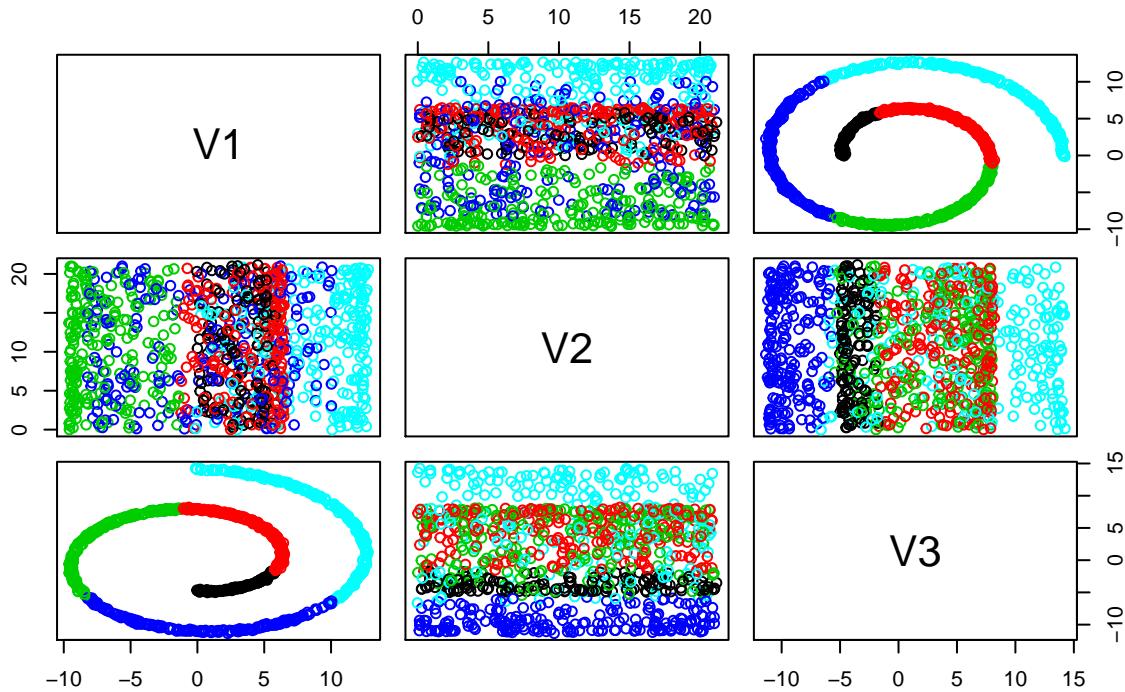
```
> scatterplot3d(hFile, col.axis = "blue", col.grid = "lightblue", main = "h.csv en 3D",  
+     pch = 20, color = hClass)
```

h.csv en 3D



```
> plot(hFile, col = hClass, main = "Archivo h.csv")
```

Archivo h.csv



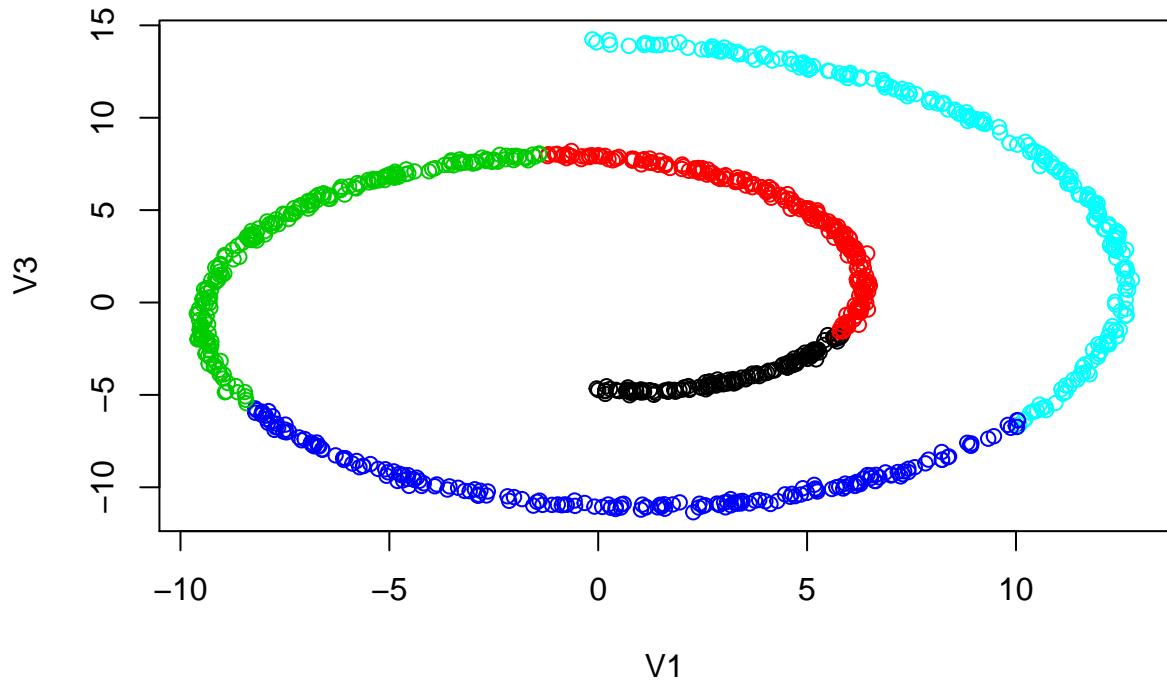
Vemos que las columnas relevantes son V_1 y V_3 , ya que con ellas se ve la figura formada por los datos (espiral).

```
> hFile = hFile[c("V1", "V3")]
> head(hFile)
```

```
##          V1          V3
## 1 -8.867581 -4.358825
## 2  5.193664 -10.064941
## 3 -6.053527 -8.423283
## 4 -8.964402 -3.891926
## 5 -6.705757  5.818809
## 6 -2.016423 -10.528826
```

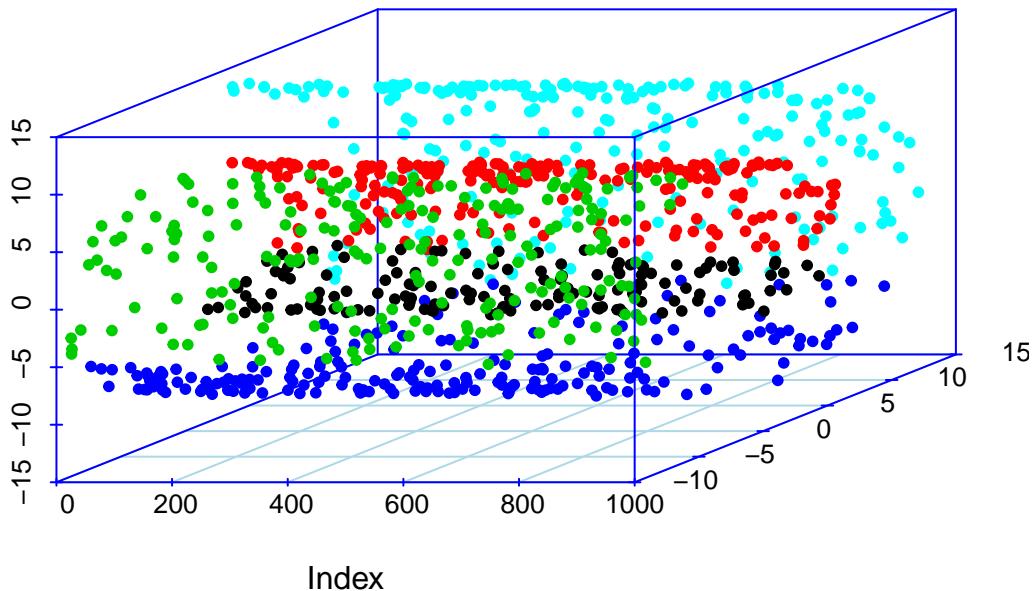
```
> plot(hFile$V1, hFile$V3, col = hClass, xlab = "V1", ylab = "V3", main = "Archivo h.csv")
```

Archivo h.csv



```
> scatterplot3d(hFile, col.axis = "blue", col.grid = "lightblue", main = "h.csv en 3D",
+     pch = 20, color = hClass)
```

h.csv en 3D

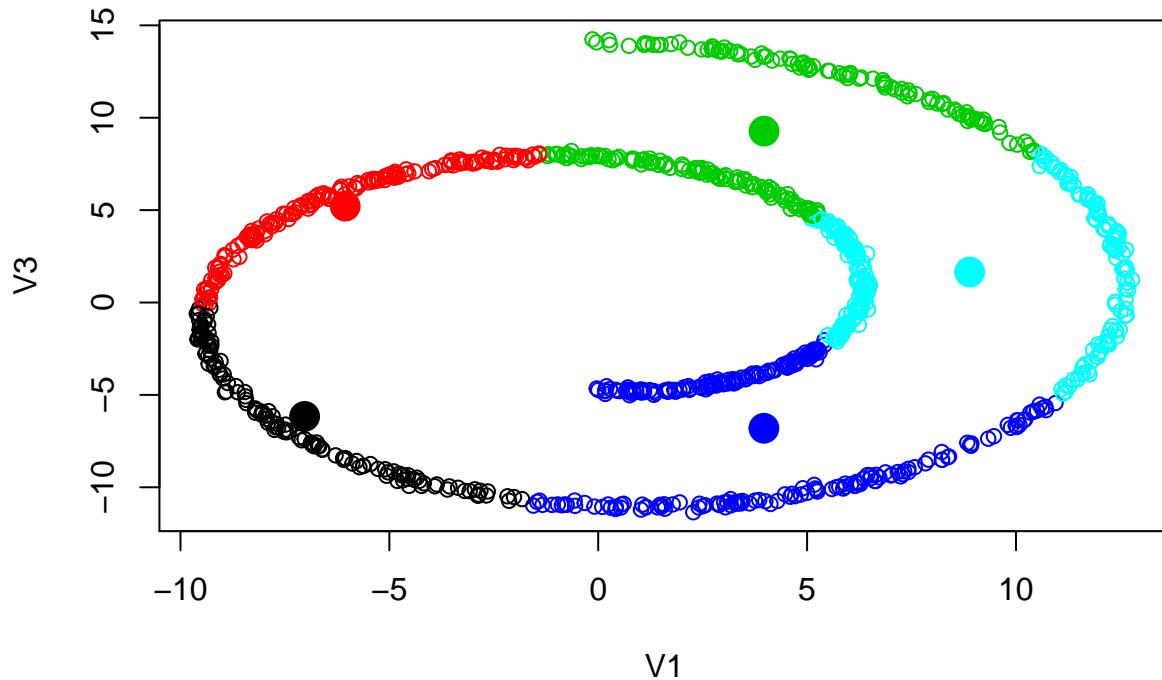


Probaremos varios métodos, para comprobar con cual se predicen mejor las clases,

5. Usando *kmeans*

```
> hFile.k.medias = kmeans(x = hFile, centers = 5)
>
> # Graficamos los clusters obtenidos en kmeans
> plot(hFile$V1, hFile$V3, col = hFile.k.medias$cluster, xlab = "V1", ylab = "V3",
+       main = "Archivo h.csv con kmedias")
>
> # Graficamos los centroides asociados a cada cluster Usamos un color
> # distinto para los centroides para verlos mejor
> points(x = hFile.k.medias$centers[, c("V1", "V3")], col = 1:5, pch = 20, cex = 3)
```

Archivo h.csv con kmedias



Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = hFile.k.medias$cluster, y = hClass, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
## |-----|  
## |           N |  
## |     N / Row Total |  
## |     N / Col Total |  
## |     N / Table Total |  
## |-----|  
##  
##  
## Total Observations in Table: 1000  
##  
##  
##          | hClass  
## hFile.k.medias$cluster |    1 |    2 |    3 |    4 |    5 | Row Total |  
## |-----|-----|-----|-----|-----|-----|-----|  
##       1 |    0 |    0 |   53 |   83 |    0 |   136 |  
## | 0.000 | 0.000 | 0.390 | 0.610 | 0.000 | 0.136 |  
## | 0.000 | 0.000 | 0.248 | 0.413 | 0.000 | |  
## | 0.000 | 0.000 | 0.053 | 0.083 | 0.000 | |  
## |-----|-----|-----|-----|-----|-----|
```

```

##          2 |      0 |      0 |    161 |      0 |      0 |      0 |    161 |
##          | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.161 |
##          | 0.000 | 0.000 | 0.752 | 0.000 | 0.000 | 0.000 |
##          | 0.000 | 0.000 | 0.161 | 0.000 | 0.000 | 0.000 |
## -----
##          3 |      0 |    115 |      0 |      0 |    105 |    220 |
##          | 0.000 | 0.523 | 0.000 | 0.000 | 0.477 | 0.220 |
##          | 0.000 | 0.525 | 0.000 | 0.000 | 0.465 | 0.000 |
##          | 0.000 | 0.115 | 0.000 | 0.000 | 0.105 | 0.000 |
## -----
##          4 |    130 |      0 |      0 |    118 |     12 |   260 |
##          | 0.500 | 0.000 | 0.000 | 0.454 | 0.046 | 0.260 |
##          | 0.929 | 0.000 | 0.000 | 0.587 | 0.053 | 0.000 |
##          | 0.130 | 0.000 | 0.000 | 0.118 | 0.012 | 0.000 |
## -----
##          5 |     10 |    104 |      0 |      0 |    109 |   223 |
##          | 0.045 | 0.466 | 0.000 | 0.000 | 0.489 | 0.223 |
##          | 0.071 | 0.475 | 0.000 | 0.000 | 0.482 | 0.000 |
##          | 0.010 | 0.104 | 0.000 | 0.000 | 0.109 | 0.000 |
## -----
##          Column Total |    140 |    219 |    214 |    201 |    226 |   1000 |
##          | 0.140 | 0.219 | 0.214 | 0.201 | 0.226 | 0.000 |
## -----
##
```

Se clasifican 260 filas de 140 como 1, 233 de 219 como 2, 135 de 214 como 3, 160 de 201 como 4 y 212 de 226 como 5.

Claramente vemos que existe un alto margen de error usando kmedias con este dataset.

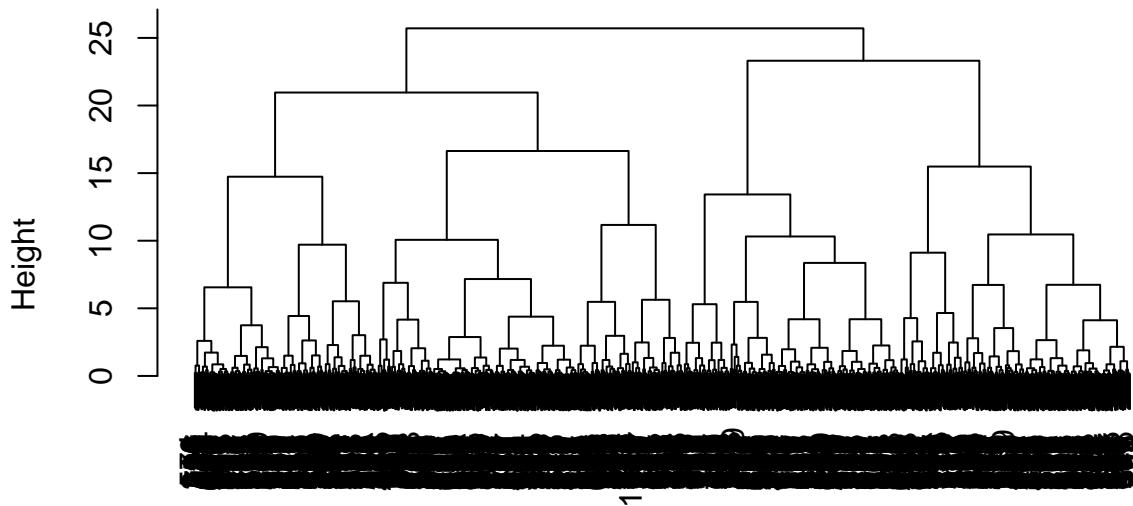
6. Usando *hclust complete*

```

> # Dataframe a Matrix
> hFileMatrix = as.matrix(hFile)
>
> # Matriz de distancia con norma 2 method = 'euclidean', 'maximum',
> # 'manhattan', 'canberra', 'binary' or 'minkowski'
> distancia = dist(hFileMatrix, method = "euclidean")
>
> # Método por defecto es complete link
> #'single', 'complete'
> metodo = "complete"
>
> cluster = hclust(distancia, method = metodo)
>
> plot(cluster, main = "Cluster jerarquico de h.csv")

```

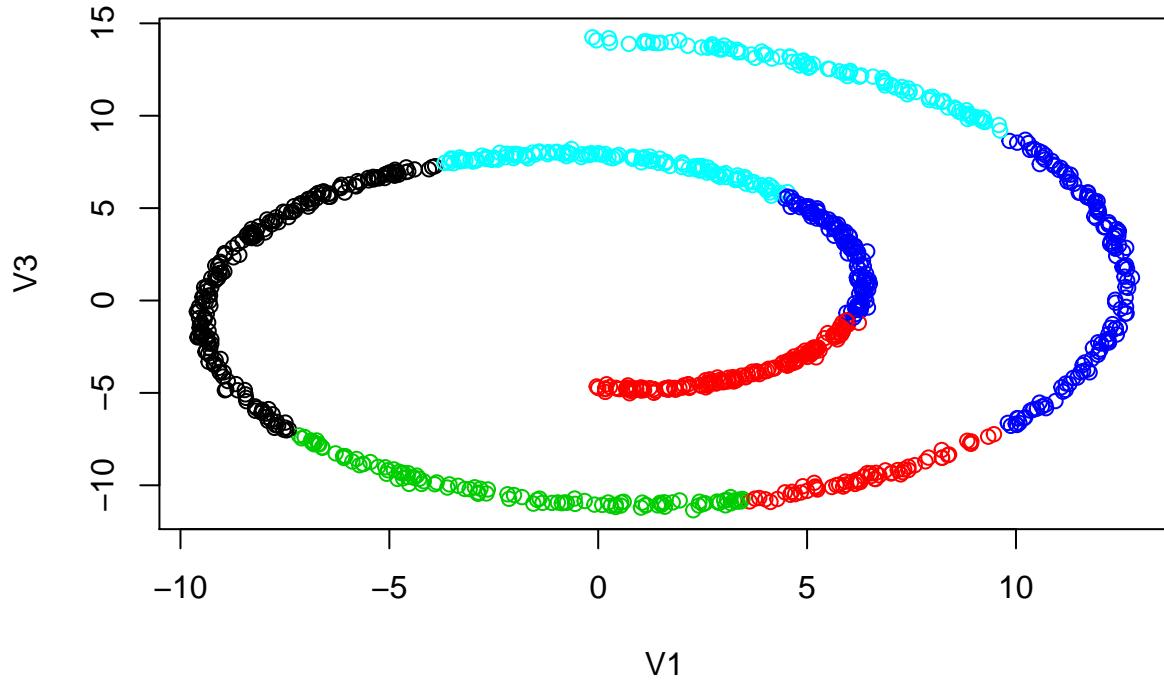
Cluster jerarquico de h.csv



distancia
hclust (*, "complete")

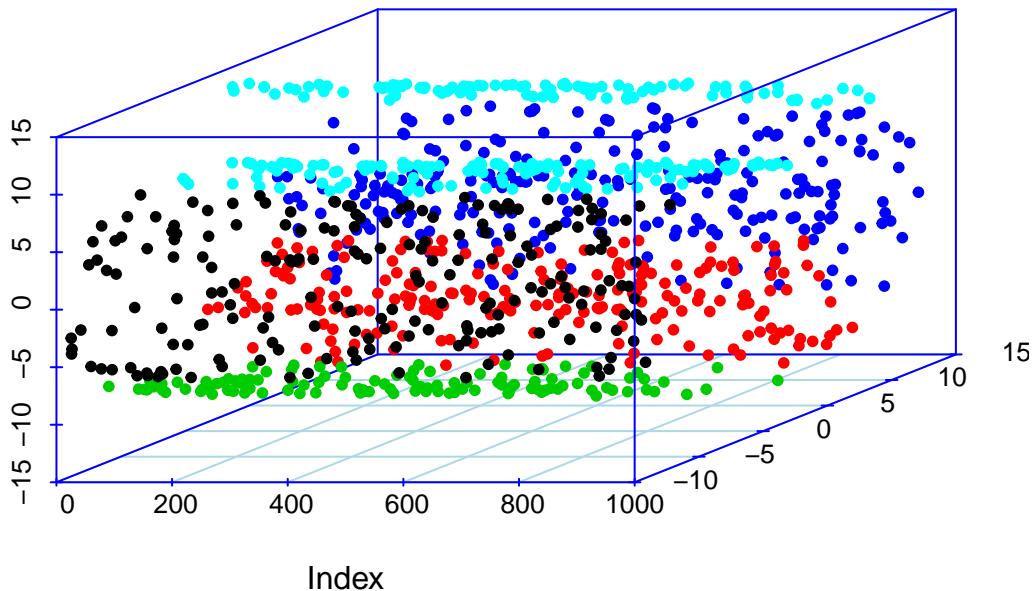
```
> # Definimos 5 clases con definir_clases()
> nclases = 5
>
> # Cortamos
> corte = cutree(cluster, k = nclases)
>
> # Graficamos el dataset con el color de las clases obtenidas con hclust
> plot(hFile$V1, hFile$V3, col = corte, xlab = "V1", ylab = "V3", main = "Archivo h.csv hclust complete")
```

Archivo h.csv hclust complete



```
> scatterplot3d(hFile, col.axis = "blue", col.grid = "lightblue", main = "h.csv en 3D",
+     pch = 20, color = corte)
```

h.csv en 3D



Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = corte, y = hClass, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
## |-----|  
## |           N |  
## |           N / Row Total |  
## |           N / Col Total |  
## |           N / Table Total |  
## |-----|  
##  
##  
## Total Observations in Table:  1000  
##  
##  
##          | hClass  
##  corte |   1 |   2 |   3 |   4 |   5 | Row Total |  
## |-----|-----|-----|-----|-----|-----|-----|  
##  1 |     0 |     0 | 174 |   21 |     0 |    195 |  
## | 0.000 | 0.000 | 0.892 | 0.108 | 0.000 | 0.195 |  
## | 0.000 | 0.000 | 0.813 | 0.104 | 0.000 | |  
## | 0.000 | 0.000 | 0.174 | 0.021 | 0.000 | |  
## |-----|-----|-----|-----|-----|-----|-----|
```

```

##      2 |     140 |      14 |       0 |      60 |       0 |     214 |
##      | 0.654 | 0.065 | 0.000 | 0.280 | 0.000 | 0.214 |
##      | 1.000 | 0.064 | 0.000 | 0.299 | 0.000 | |
##      | 0.140 | 0.014 | 0.000 | 0.060 | 0.000 | |
## -----
##      3 |      0 |      0 |       0 |     114 |       0 |    114 |
##      | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.114 |
##      | 0.000 | 0.000 | 0.000 | 0.567 | 0.000 | |
##      | 0.000 | 0.000 | 0.000 | 0.114 | 0.000 | |
## -----
##      4 |      0 |     112 |       0 |      6 |    128 |    246 |
##      | 0.000 | 0.455 | 0.000 | 0.024 | 0.520 | 0.246 |
##      | 0.000 | 0.511 | 0.000 | 0.030 | 0.566 | |
##      | 0.000 | 0.112 | 0.000 | 0.006 | 0.128 | |
## -----
##      5 |      0 |     93 |     40 |       0 |     98 |    231 |
##      | 0.000 | 0.403 | 0.173 | 0.000 | 0.424 | 0.231 |
##      | 0.000 | 0.425 | 0.187 | 0.000 | 0.434 | |
##      | 0.000 | 0.093 | 0.040 | 0.000 | 0.098 | |
## -----
## Column Total |     140 |     219 |     214 |     201 |     226 |    1000 |
##      | 0.140 | 0.219 | 0.214 | 0.201 | 0.226 | |
## -----
##
```

Se clasifican 195 filas de 140 como 1, 214 de 219 como 2, 114 de 214 como 3, 246 de 201 como 4 y 231 de 226 como 5.

La clasificación sigue siendo mala por el margen de error que existe.

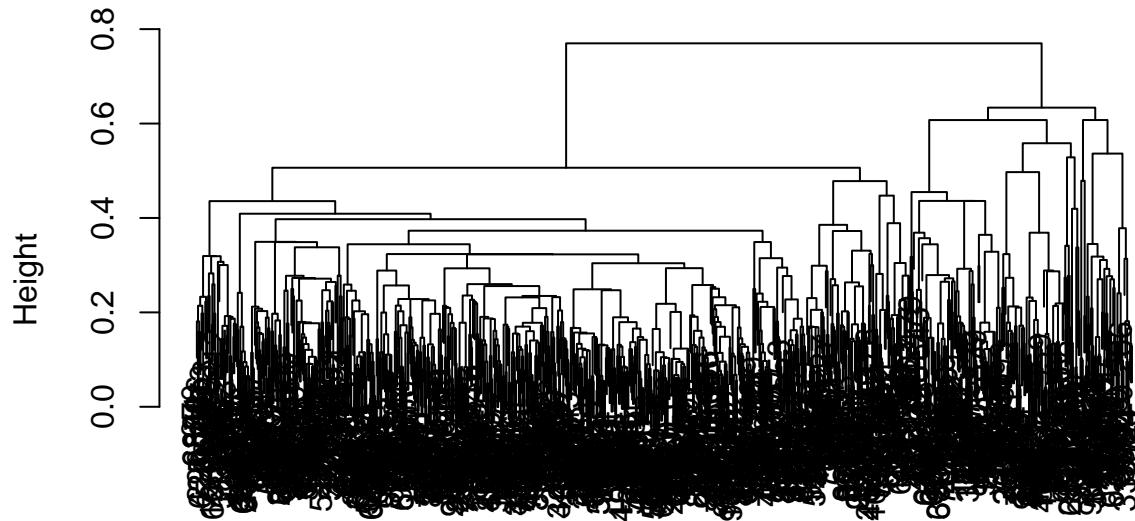
7. Usando *hclust single*

```

> # Dataframe a Matrix
> hFileMatrix = as.matrix(hFile)
>
> # Matriz de distancia con norma 2 method = 'euclidean', 'maximum',
> # 'manhattan', 'canberra', 'binary' or 'minkowski'
> distancia = dist(hFileMatrix, method = "euclidean")
>
> # Método por defecto es complete link
> #'single', 'complete'
> metodo = "single"
>
> cluster = hclust(distancia, method = metodo)
>
> plot(cluster, main = "Cluster jerárquico de h.csv")

```

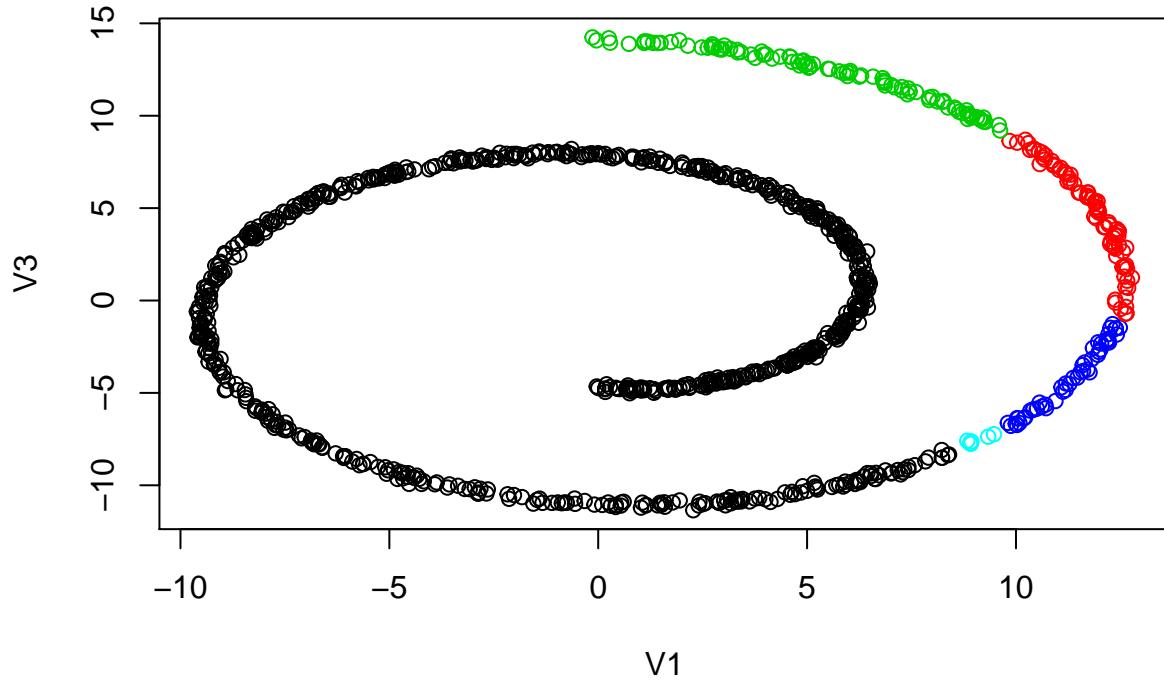
Cluster jerarquico de h.csv



distancia
hclust (*, "single")

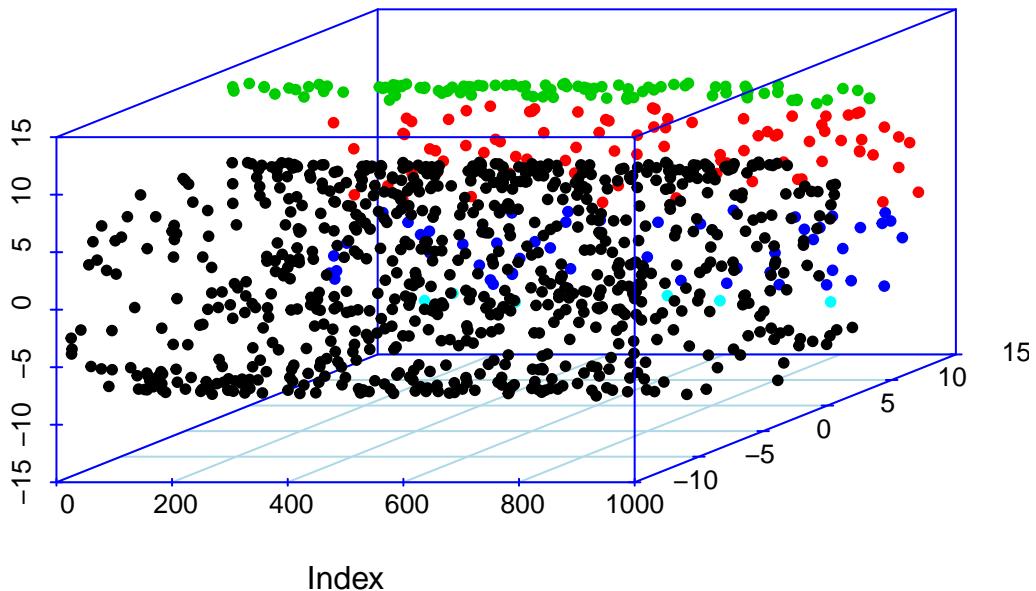
```
> # Definimos 5 clases con definir_clases()  
> nclases = 5  
>  
> # Cortamos  
> corte = cutree(cluster, k = nclases)  
>  
> # Graficamos el dataset con el color de las clases obtenidas con hclust  
> plot(hFile$V1, hFile$V3, col = corte, xlab = "V1", ylab = "V3", main = "Archivo h.csv hclust completo")
```

Archivo h.csv hclust complete



```
> scatterplot3d(hFile, col.axis = "blue", col.grid = "lightblue", main = "h.csv en 3D",
+     pch = 20, color = corte)
```

h.csv en 3D



Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = corte, y = hClass, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
## |-----|  
## |           N |  
## |     N / Row Total |  
## |     N / Col Total |  
## |     N / Table Total |  
## |-----|  
##  
##  
## Total Observations in Table: 1000  
##  
##  
##          | hClass  
##   corte |    1 |    2 |    3 |    4 |    5 | Row Total |  
## |-----|-----|-----|-----|-----|-----|-----|  
##   1 |    140 |   219 |   214 |   189 |    0 |    762 |  
## | 0.184 | 0.287 | 0.281 | 0.248 | 0.000 | 0.762 |  
## | 1.000 | 1.000 | 1.000 | 0.940 | 0.000 | |  
## | 0.140 | 0.219 | 0.214 | 0.189 | 0.000 | |  
## |-----|-----|-----|-----|-----|-----|-----|
```

```

##      2 |      0 |      0 |      0 |      0 |      84 |      84 |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.084 |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 0.372 | |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 0.084 | |
## -----
##      3 |      0 |      0 |      0 |      0 |      98 |      98 |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.098 |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 0.434 | |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 0.098 | |
## -----
##      4 |      0 |      0 |      0 |      6 |      44 |      50 |
##      | 0.000 | 0.000 | 0.000 | 0.120 | 0.880 | 0.050 |
##      | 0.000 | 0.000 | 0.000 | 0.030 | 0.195 | |
##      | 0.000 | 0.000 | 0.000 | 0.006 | 0.044 | |
## -----
##      5 |      0 |      0 |      0 |      6 |      0 |      6 |
##      | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.006 |
##      | 0.000 | 0.000 | 0.000 | 0.030 | 0.000 | |
##      | 0.000 | 0.000 | 0.000 | 0.006 | 0.000 | |
## -----
## Column Total | 140 | 219 | 214 | 201 | 226 | 1000 |
##      | 0.140 | 0.219 | 0.214 | 0.201 | 0.226 | |
## -----
## 
```

Se clasifican 762 filas de 140 como 1, 84 de 219 como 2, 114 de 98 como 3, 246 de 50 como 4 y 6 de 226 como 5.

Con este método la clasificación es peor que con los anteriores.

8. Conclusión:

Como a mencionamos usando cluster jerárquico con *single link* obtenemos los peores resultados. Con k medias existe un alto margen de error, al igual que con cluster jerárquico con *complete link*, pero en este último la clasificación es ligeramente mejor.

En general, ninguno de los métodos usados funcionan de forma correcta con este dataset debido a su forma en espiral. Se debe usar un método que se ajuste de manera correcta a esta forma.

s.csv

1. Cargamos el dataset *s.csv*, asumiendo que se hizo **setwd(directorio)**:

```

> sFile = read.csv(file = "./s.csv", header = FALSE, sep = ",")
> head(sFile)

```

```

##      V1      V2      V3      V4
## 1  0.443999 1.185761 -0.103973  0.460056
## 2  0.897241 0.020127 -1.441541  2.028112
## 3  0.824049 0.951652 -0.433482  0.968522
## 4  0.410511 1.417541 -0.088144  0.423014
## 5 -0.659036 0.087951  0.247889 -0.719537
## 6  0.980897 1.759043 -0.805472  1.375020

```

```
> summary(sFile$V4)

##      Min.    1st Qu.     Median      Mean   3rd Qu.      Max.
## -4.70700 -2.37900 -0.17600 -0.03844  2.23600  4.71100
```

2. La última columna de *s.csv* contiene las clases asociadas a las instancias del dataset, pero estos valores se encuentran en el intervalo (-4,5) por lo que debemos transformarlas y para esto implementamos la función *definir_clase*

```
> definir_clase_s = function(numero) {
+   if (numero <= -3)
+     return(1) else if (numero > -3 && numero <= -1)
+     return(2) else if (numero > -1 && numero <= 1)
+     return(3) else if (numero > 1 && numero <= 3)
+     return(4) else return(5)
+ }
```

Ahora, usamos esta función para asignar una clase a cada instancia del dataset,

```
> for (i in 1:length(sFile$V4)) {
+   sFile$V4[i] = definir_clase_s(sFile$V4[i])
+ }
>
> # sFile$V4
```

3. Separamos la última columna del dataset. La cual, hace referencia a la clase de cada instancia,

```
> sClass = sFile$V4
> sFile = sFile[c("V1", "V2", "V3")]
> head(sFile)
```

```
##           V1          V2          V3
## 1  0.443999 1.185761 -0.103973
## 2  0.897241 0.020127 -1.441541
## 3  0.824049 0.951652 -0.433482
## 4  0.410511 1.417541 -0.088144
## 5 -0.659036 0.087951  0.247889
## 6  0.980897 1.759043 -0.805472
```

Vemos cuantas clases contiene el dataset

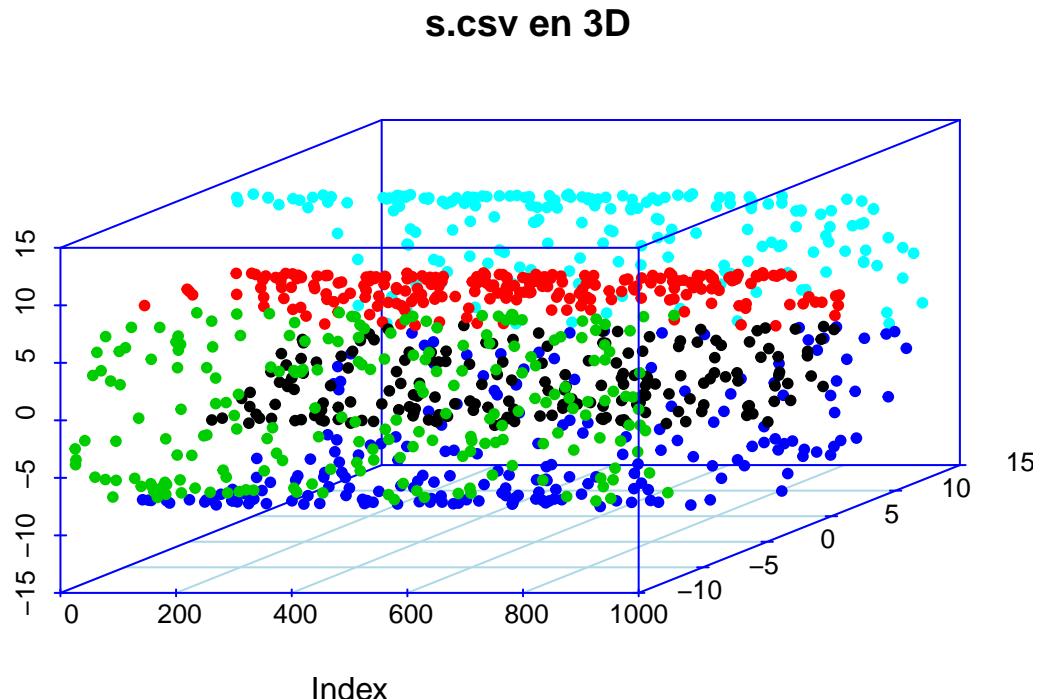
```
> table(sClass)
```

```
## sClass
##  1  2  3  4  5
## 185 220 211 197 187
```

El dataset posee 5 clases.

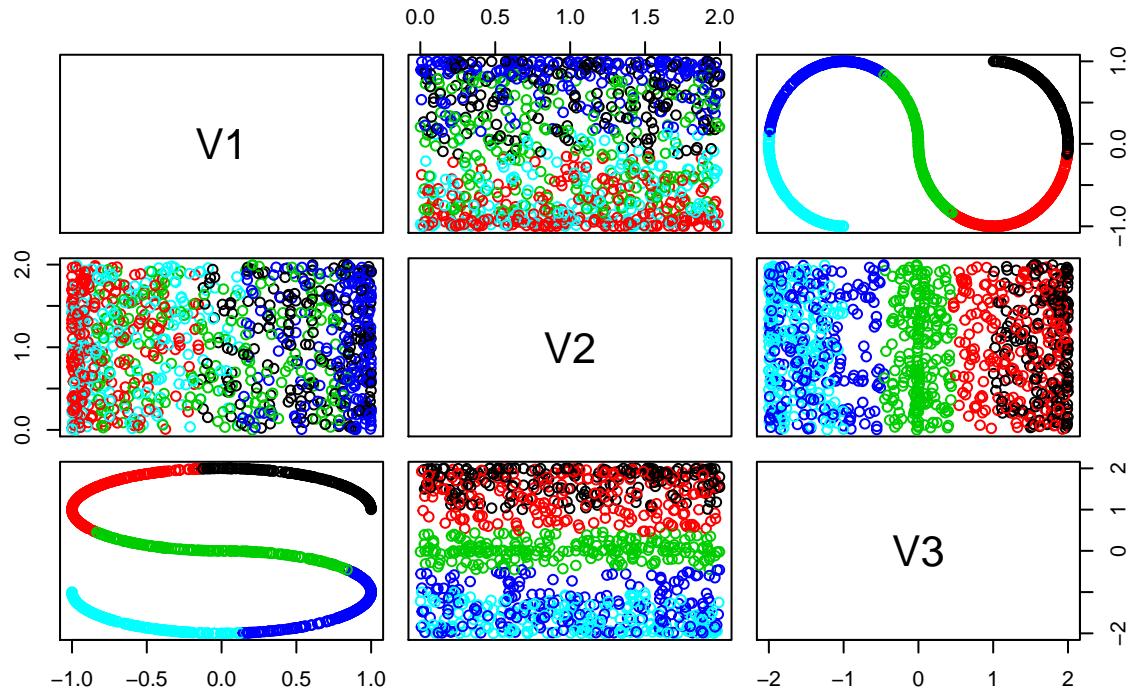
4. Ahora graficamos el dataset, para verificar como se comportan los datos

```
> scatterplot3d(hFile, col.axis = "blue", col.grid = "lightblue", main = "s.csv en 3D",
+     pch = 20, color = sClass)
```



```
> plot(sFile, col = sClass, main = "Archivo s.csv")
```

Archivo s.csv



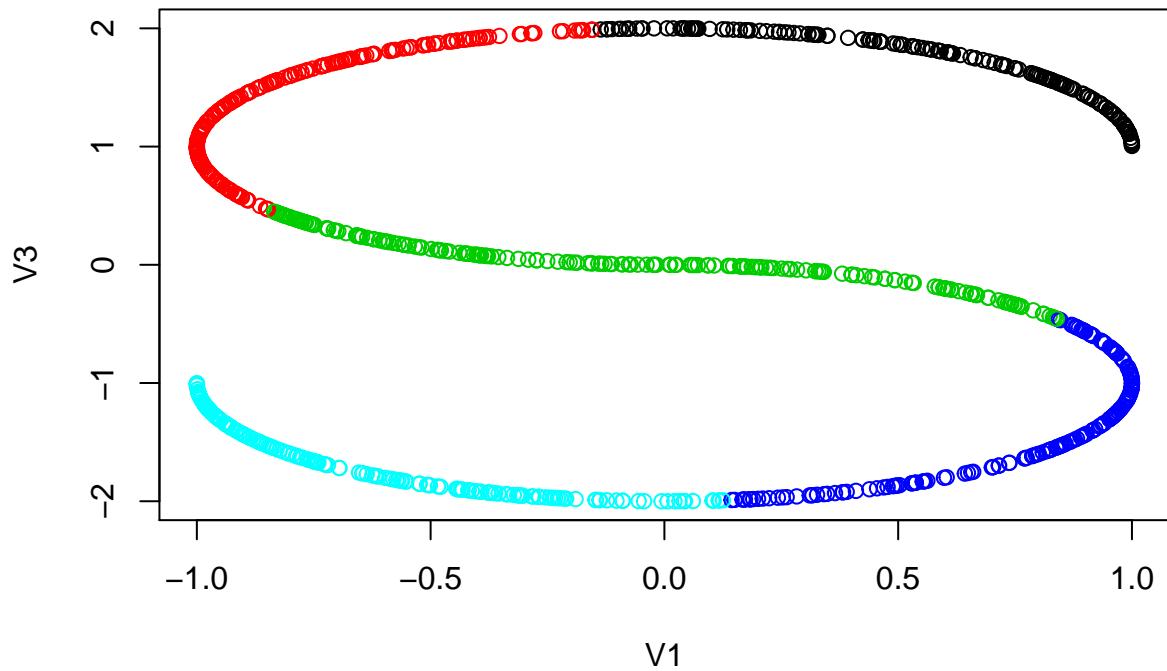
Vemos que las columnas relevantes son V_1 y V_3 , ya que con ellas se ve la figura formada por los datos (S).

```
> sFile = sFile[c("V1", "V3")]
> head(sFile)
```

```
##          V1         V3
## 1  0.443999 -0.103973
## 2  0.897241 -1.441541
## 3  0.824049 -0.433482
## 4  0.410511 -0.088144
## 5 -0.659036  0.247889
## 6  0.980897 -0.805472
```

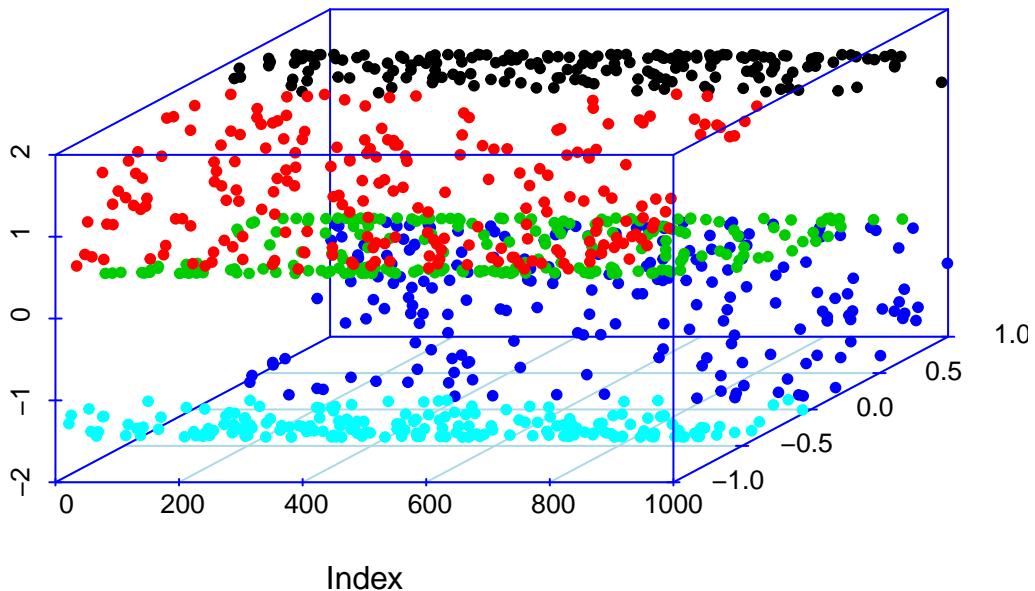
```
> plot(sFile$V1, sFile$V3, col = sClass, xlab = "V1", ylab = "V3", main = "Archivo s.csv")
```

Archivo s.csv



```
> scatterplot3d(sFile, col.axis = "blue", col.grid = "lightblue", main = "s.csv en 3D",
+     pch = 20, color = sClass)
```

s.csv en 3D

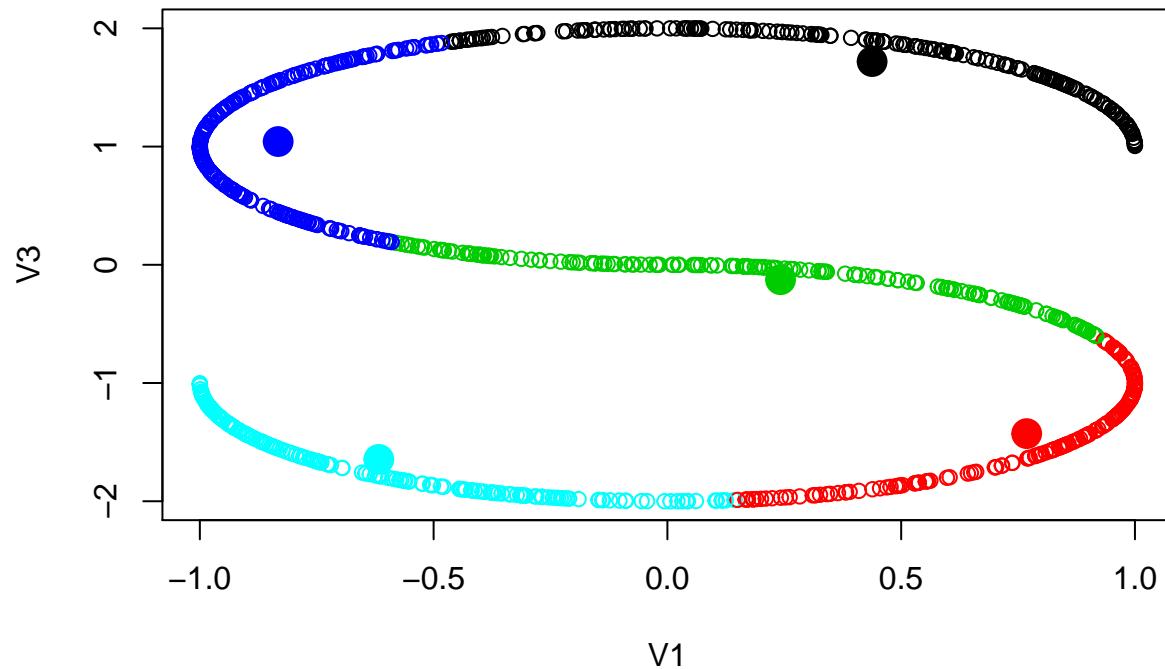


Probaremos varios métodos, para comprobar con cual se predicen mejor las clases,

5. Usando *kmeans*

```
> sFile.k.medias = kmeans(x = sFile, centers = 5)
>
> # Graficamos los clusters obtenidos en kmeans
> plot(sFile$V1, sFile$V3, col = sFile.k.medias$cluster, xlab = "V1", ylab = "V3",
+       main = "Archivo s.csv con kmedias")
>
> # Graficamos los centroides asociados a cada cluster Usamos un color
> # distinto para los centroides para verlos mejor
> points(x = sFile.k.medias$centers[, c("V1", "V3")], col = 1:5, pch = 20, cex = 3)
```

Archivo s.csv con kmedias



Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = sFile.k.medias$cluster, y = sClass, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
##  |-----|  
##  |           N |  
##  |   N / Row Total |  
##  |   N / Col Total |  
##  |   N / Table Total |  
##  |-----|  
##  
##  
##  Total Observations in Table:  1000  
##  
##  
##  
##          | sClass  
## sFile.k.medias$cluster |    1 |    2 |    3 |    4 |    5 | Row Total |  
##  |-----|-----|-----|-----|-----|-----|-----|  
##  1 |    185 |    32 |     0 |     0 |     0 |    217 |  
##  |  0.853 |  0.147 |  0.000 |  0.000 |  0.000 |  0.217 |  
##  |  1.000 |  0.145 |  0.000 |  0.000 |  0.000 |  |  
##  |  0.185 |  0.032 |  0.000 |  0.000 |  0.000 |  |  
##  |-----|-----|-----|-----|-----|-----|
```

```

##          2 |      0 |      0 |      0 |     174 |      0 |      0 |     174 |
##          | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.174 |
##          | 0.000 | 0.000 | 0.000 | 0.883 | 0.000 | 0.000 | |
##          | 0.000 | 0.000 | 0.000 | 0.174 | 0.000 | 0.000 | |
## -----
##          3 |      0 |      0 |    161 |     22 |      0 |      0 |    183 |
##          | 0.000 | 0.000 | 0.880 | 0.120 | 0.000 | 0.000 | 0.183 |
##          | 0.000 | 0.000 | 0.763 | 0.112 | 0.000 | 0.000 | |
##          | 0.000 | 0.000 | 0.161 | 0.022 | 0.000 | 0.000 | |
## -----
##          4 |      0 |    188 |      50 |      0 |      0 |      0 |    238 |
##          | 0.000 | 0.790 | 0.210 | 0.000 | 0.000 | 0.000 | 0.238 |
##          | 0.000 | 0.855 | 0.237 | 0.000 | 0.000 | 0.000 | |
##          | 0.000 | 0.188 | 0.050 | 0.000 | 0.000 | 0.000 | |
## -----
##          5 |      0 |      0 |      0 |      1 |     187 |     188 |
##          | 0.000 | 0.000 | 0.000 | 0.005 | 0.995 | 0.188 |
##          | 0.000 | 0.000 | 0.000 | 0.005 | 1.000 | 1.000 |
##          | 0.000 | 0.000 | 0.000 | 0.001 | 0.187 | 0.187 |
## -----
##          Column Total |    185 |    220 |    211 |    197 |    187 |    1000 |
##          | 0.185 | 0.220 | 0.211 | 0.197 | 0.187 | 0.187 |
## -----
##          |
##          |

```

Existe un margen de error que no es tan grande como el del dataset anterior.

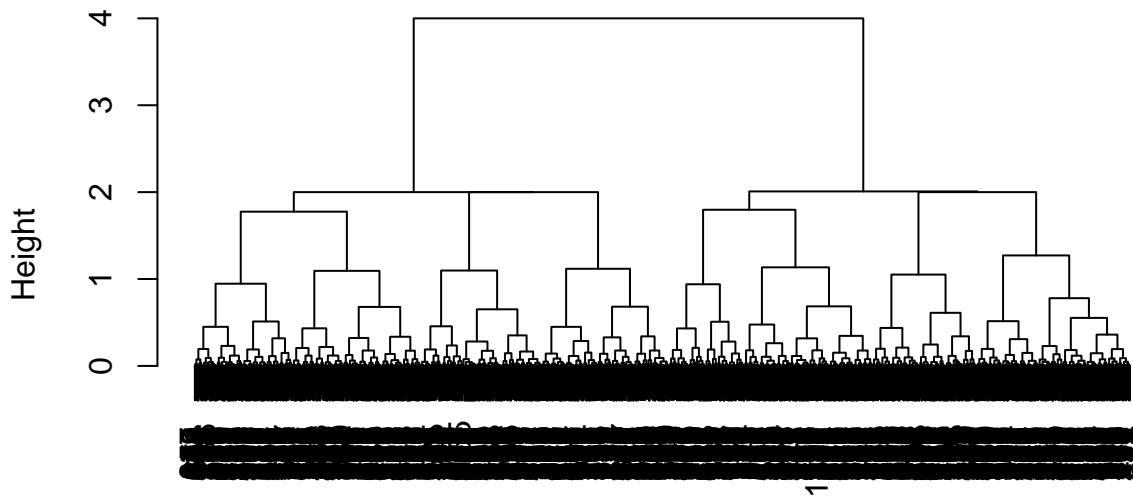
6. Usando *hclust complete*

```

> # Dataframe a Matrix
> sFileMatrix = as.matrix(sFile)
>
> # Matriz de distancia con norma 2 method = 'euclidean', 'maximum',
> # 'manhattan', 'canberra', 'binary' or 'minkowski'
> distancia = dist(sFileMatrix, method = "euclidean")
>
> # Método por defecto es complete link
> #'single', 'complete'
> metodo = "complete"
>
> cluster = hclust(distancia, method = metodo)
>
> plot(cluster, main = "Cluster jerarquico de s.csv")

```

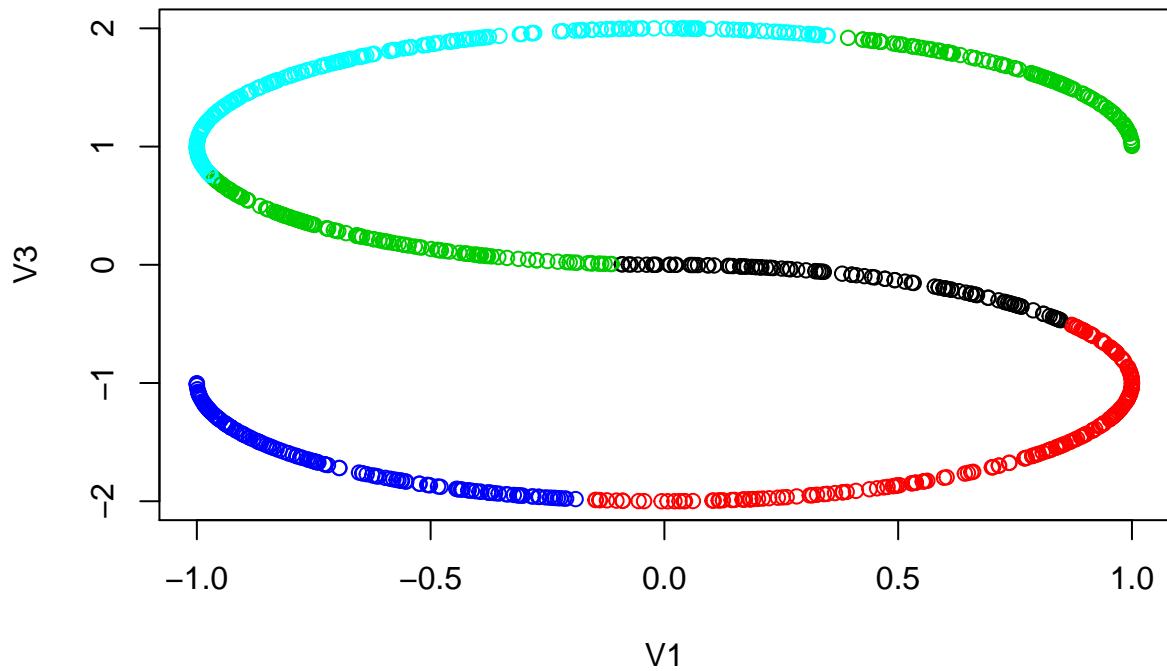
Cluster jerarquico de s.csv



distancia
hclust (*, "complete")

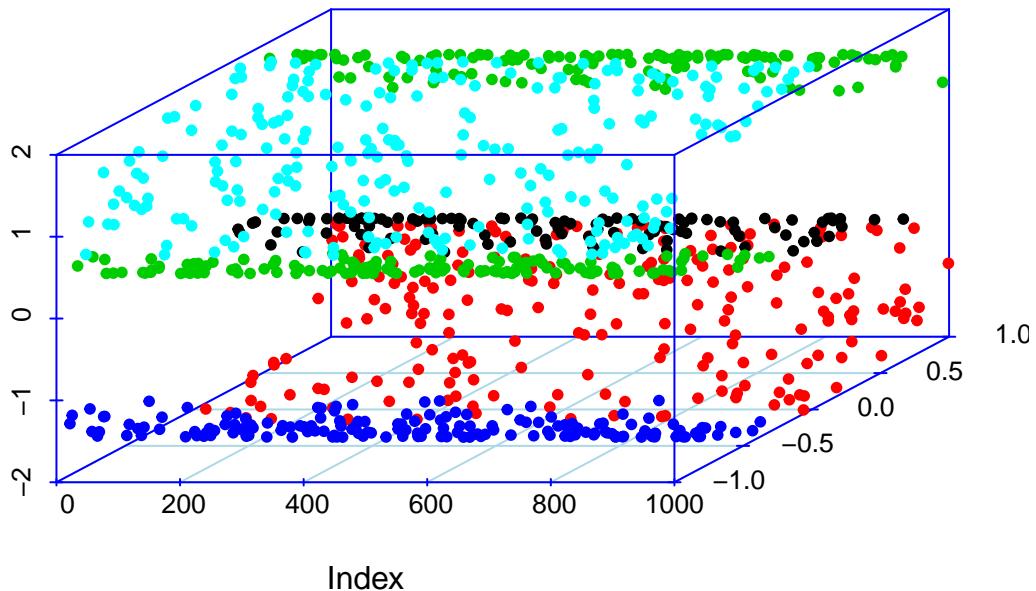
```
> # Definimos 5 clases con definir_clases()  
> nclases = 5  
>  
> # Cortamos  
> corte = cutree(cluster, k = nclases)  
>  
> # Graficamos el dataset con el color de las clases obtenidas con hclust  
> plot(sFile$V1, sFile$V3, col = corte, xlab = "V1", ylab = "V3", main = "Archivo s.csv hclust complete")
```

Archivo s.csv hclust complete



```
> scatterplot3d(sFile, col.axis = "blue", col.grid = "lightblue", main = "s.csv en 3D",
+     pch = 20, color = corte)
```

s.csv en 3D



Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = corte, y = sClass, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
## |-----|  
## |           N |  
## |           N / Row Total |  
## |           N / Col Total |  
## |           N / Table Total |  
## |-----|  
##  
##  
## Total Observations in Table: 1000  
##  
##  
##          | sClass  
##   corte |    1 |    2 |    3 |    4 |    5 | Row Total |  
## -----|-----|-----|-----|-----|-----|-----|  
##   1 |     0 |     0 | 108 |     3 |     0 |    111 |  
## | 0.000 | 0.000 | 0.973 | 0.027 | 0.000 | 0.111 |  
## | 0.000 | 0.000 | 0.512 | 0.015 | 0.000 | |  
## | 0.000 | 0.000 | 0.108 | 0.003 | 0.000 | |  
## -----|-----|-----|-----|-----|-----|-----|
```

```

##      2 |      0 |      0 |      0 |    194 |     22 |    216 |
##      | 0.000 | 0.000 | 0.000 | 0.898 | 0.102 | 0.216 |
##      | 0.000 | 0.000 | 0.000 | 0.985 | 0.118 |
##      | 0.000 | 0.000 | 0.000 | 0.194 | 0.022 |
## -----
##      3 | 129 | 34 | 103 | 0 | 0 | 266 |
##      | 0.485 | 0.128 | 0.387 | 0.000 | 0.000 | 0.266 |
##      | 0.697 | 0.155 | 0.488 | 0.000 | 0.000 |
##      | 0.129 | 0.034 | 0.103 | 0.000 | 0.000 |
## -----
##      4 | 0 | 0 | 0 | 0 | 165 | 165 |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.165 |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 0.882 |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 0.165 |
## -----
##      5 | 56 | 186 | 0 | 0 | 0 | 242 |
##      | 0.231 | 0.769 | 0.000 | 0.000 | 0.000 | 0.242 |
##      | 0.303 | 0.845 | 0.000 | 0.000 | 0.000 |
##      | 0.056 | 0.186 | 0.000 | 0.000 | 0.000 |
## -----
## Column Total | 185 | 220 | 211 | 197 | 187 | 1000 |
##      | 0.185 | 0.220 | 0.211 | 0.197 | 0.187 |
## -----
##
```

Se clasifican 762 instancias en la clase 1, cuando en verdad solo hay 185 en dicha clase. Vemos que el margen de error aumenta considerablemente con respecto al método anterior.

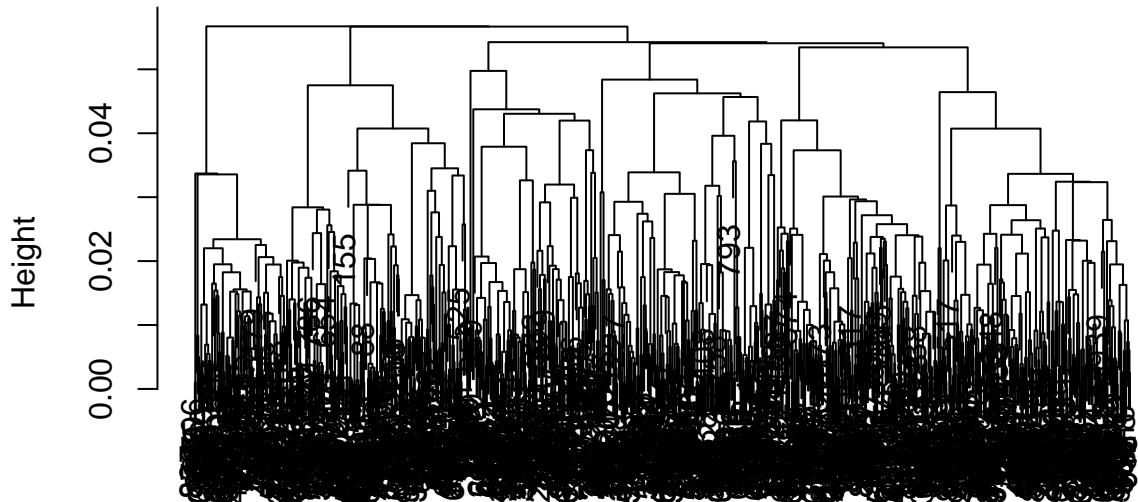
7. Usando *hclust single*

```

> # Dataframe a Matrix
> sFileMatrix = as.matrix(sFile)
>
> # Matriz de distancia con norma 2 method = 'euclidean', 'maximum',
> # 'manhattan', 'canberra', 'binary' or 'minkowski'
> distancia = dist(sFileMatrix, method = "euclidean")
>
> # Método por defecto es complete link
> #'single', 'complete'
> metodo = "single"
>
> cluster = hclust(distancia, method = metodo)
>
> plot(cluster, main = "Cluster jerarquico de s.csv")

```

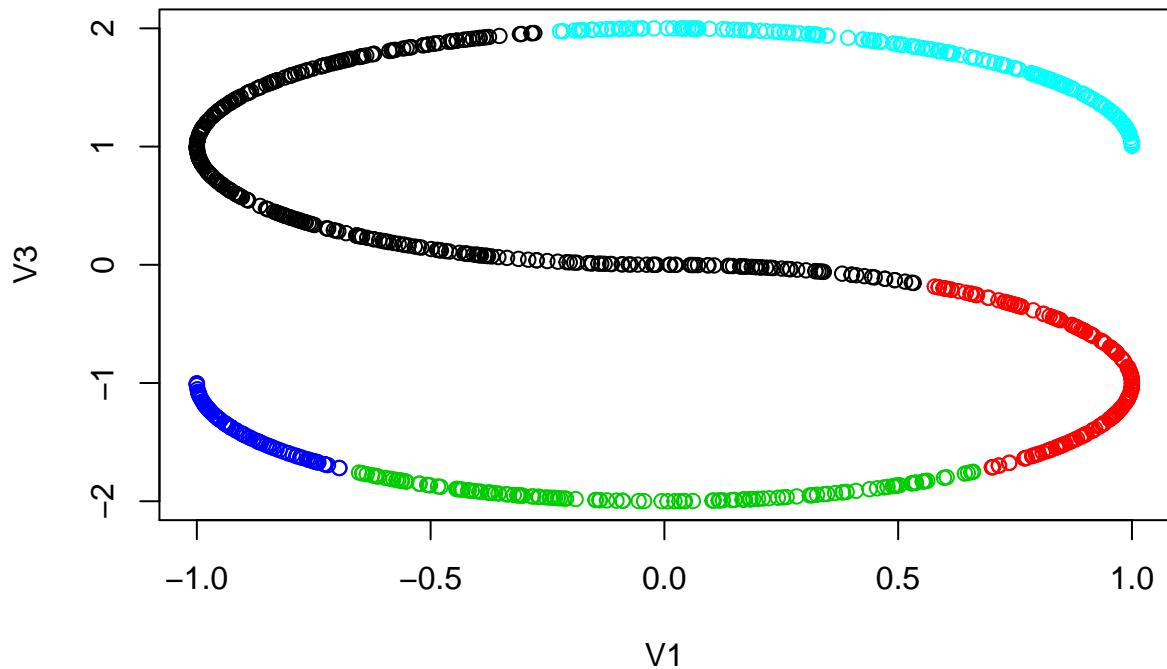
Cluster jerarquico de s.csv



distancia
hclust (*, "single")

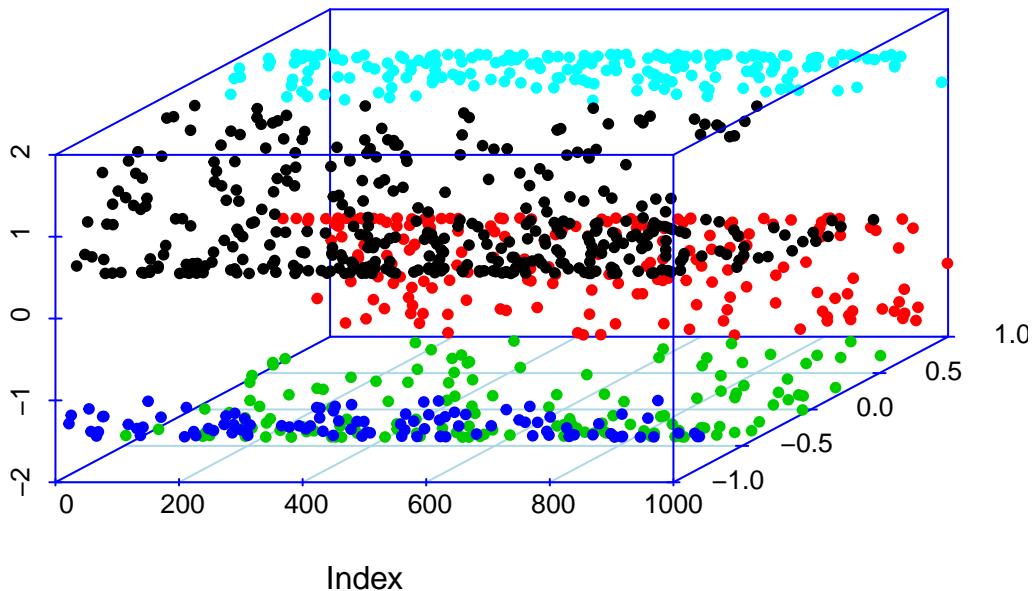
```
> # Definimos 5 clases con definir_clases()
> nclases = 5
>
> # Cortamos
> corte = cutree(cluster, k = nclases)
>
> # Graficamos el dataset con el color de las clases obtenidas con hclust
> plot(sFile$V1, sFile$V3, col = corte, xlab = "V1", ylab = "V3", main = "Archivo s.csv hclust complete")
```

Archivo s.csv hclust complete



```
> scatterplot3d(sFile, col.axis = "blue", col.grid = "lightblue", main = "s.csv en 3D",
+     pch = 20, color = corte)
```

s.csv en 3D



Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = corte, y = sClass, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
## |-----|  
## |           N |  
## |           N / Row Total |  
## |           N / Col Total |  
## |           N / Table Total |  
## |-----|  
##  
##  
## Total Observations in Table: 1000  
##  
##  
##          | sClass  
##   corte |    1 |    2 |    3 |    4 |    5 | Row Total |  
## -----|-----|-----|-----|-----|-----|-----|  
##   1 |     0 | 210 | 168 |     0 |     0 | 378 |  
## | 0.000 | 0.556 | 0.444 | 0.000 | 0.000 | 0.378 |  
## | 0.000 | 0.955 | 0.796 | 0.000 | 0.000 | |  
## | 0.000 | 0.210 | 0.168 | 0.000 | 0.000 | |  
## -----|-----|-----|-----|-----|-----|-----|
```

```

##      2 |      0 |      0 |      43 |     146 |      0 |     189 |
##      | 0.000 | 0.000 | 0.228 | 0.772 | 0.000 | 0.189 |
##      | 0.000 | 0.000 | 0.204 | 0.741 | 0.000 |
##      | 0.000 | 0.000 | 0.043 | 0.146 | 0.000 |
## -----
##      3 |      0 |      0 |      0 |      51 |      89 |     140 |
##      | 0.000 | 0.000 | 0.000 | 0.364 | 0.636 | 0.140 |
##      | 0.000 | 0.000 | 0.000 | 0.259 | 0.476 |
##      | 0.000 | 0.000 | 0.000 | 0.051 | 0.089 |
## -----
##      4 |      0 |      0 |      0 |      0 |      98 |      98 |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.098 |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 0.524 |
##      | 0.000 | 0.000 | 0.000 | 0.000 | 0.098 |
## -----
##      5 |    185 |     10 |      0 |      0 |      0 |     195 |
##      | 0.949 | 0.051 | 0.000 | 0.000 | 0.000 | 0.195 |
##      | 1.000 | 0.045 | 0.000 | 0.000 | 0.000 |
##      | 0.185 | 0.010 | 0.000 | 0.000 | 0.000 |
## -----
## Column Total |    185 |    220 |    211 |    197 |    187 |   1000 |
##      | 0.185 | 0.220 | 0.211 | 0.197 | 0.187 |
## -----
##
```

De nuevo, existen muchos errores en la clasificación usando este método.

8. Conclusión:

Como mencionamos usando cluster jerárquico con *complete link* obtenemos los peores resultados. Con cluster jerárquico con *complete link* existe un alto margen de error y con K Medias se obtienen resultados aceptables, considerando que la forma de los datos no es cilíndrica.

En general, ninguno de los métodos usados funcionan de forma correcta con este dataset debido a su forma en “S”. Se debe usar un método que se ajuste de manera correcta a esta forma.

help.csv

1. Cargamos el dataset *help.csv*, asumiendo que se hizo `setwd(directorio)`:

```

> helpFile = read.csv(file = "./help.csv", header = FALSE, sep = ",")  

> head(helpFile)

```

```

##      V1        V2        V3        V4
## 1  5.047159 17.420820 18.632855 -3.670645
## 2 -8.961992  0.774681  5.563481 -1.111127
## 3  6.534056 16.181083 -2.429920  0.712075
## 4  0.345659 19.325033 -0.005976  0.034573
## 5  3.074971 11.407641 19.515490 -3.454154
## 6  5.144209 10.541402 18.575378 -3.681925

```

```

> summary(helpFile$V4)

##      Min.   1st Qu.   Median   Mean 3rd Qu.   Max.
## -4.712  -1.180    2.426   3.141   7.045  14.140

```

2. La última columna de *help.csv* contiene las clases asociadas a las instancias del dataset, pero estos valores se encuentran en el intervalo (-4,15) por lo que debemos transformarlas y para esto implementamos la función *definir_clase*

```

> definir_clase_help = function(numero) {
+   if (numero <= -1)
+     return(1) else if (numero > -1 && numero < 4.7188)
+     return(2) else return(3)
+ }

```

Ahora, usamos esta función para asignar una clase a cada instancia del dataset,

```

> for (i in 1:length(helpFile$V4)) {
+   helpFile$V4[i] = definir_clase_help(helpFile$V4[i])
+ }
>
> # helpFile$V4

```

3. Separamos la última columna del dataset. La cual, hace referencia a la clase de cada instancia,

```

> helpClass = helpFile$V4
> helpFile = helpFile[c("V1", "V2", "V3")]
> head(helpFile)

```

```

##           V1          V2          V3
## 1  5.047159 17.420820 18.632855
## 2 -8.961992  0.774681  5.563481
## 3  6.534056 16.181083 -2.429920
## 4  0.345659 19.325033 -0.005976
## 5  3.074971 11.407641 19.515490
## 6  5.144209 10.541402 18.575378

```

Vemos cuantas clases contiene el dataset

```

> table(helpClass)

```

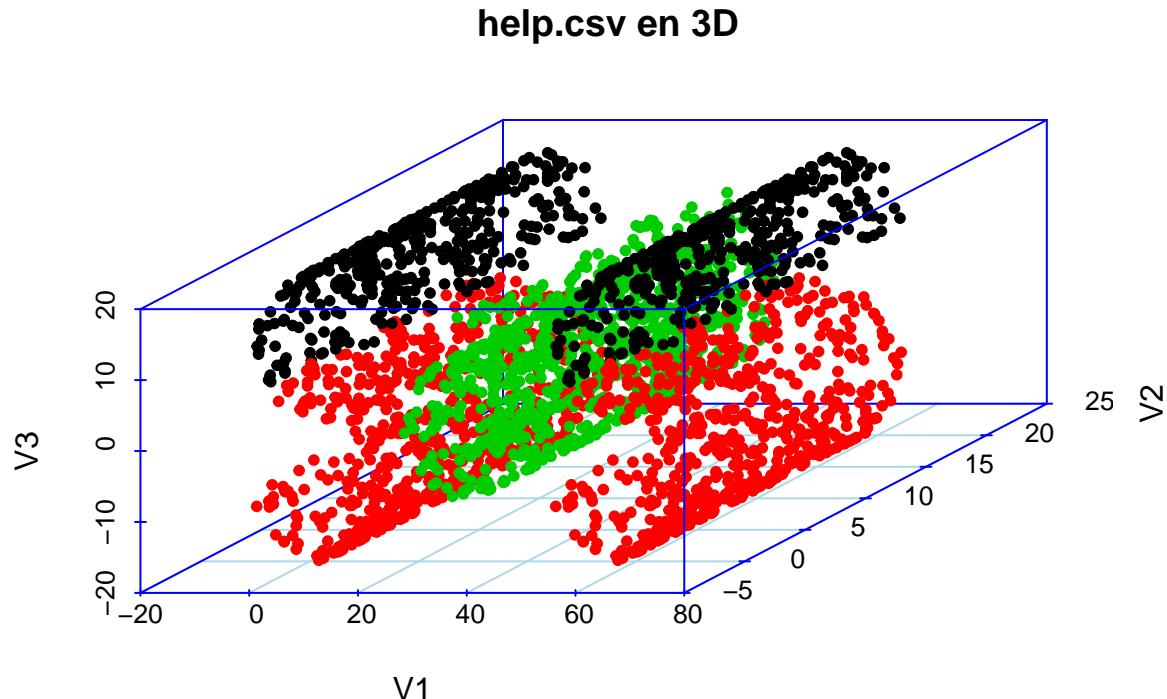
```

## helpClass
##   1   2   3
## 796 1205 999

```

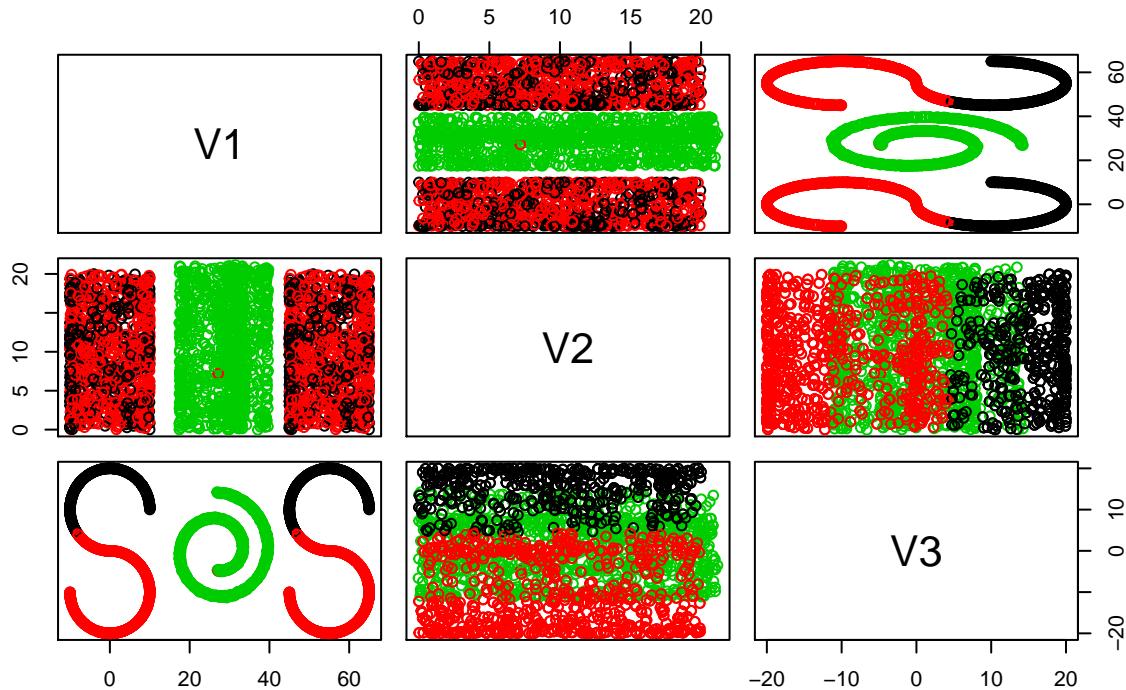
4. Ahora graficamos el dataset, para verificar como se comportan los datos

```
> scatterplot3d(helpFile, col.axis = "blue", col.grid = "lightblue", main = "help.csv en 3D",
+     pch = 20, color = helpClass)
```



```
> plot(helpFile, col = helpClass, main = "Archivo help.csv")
```

Archivo help.csv



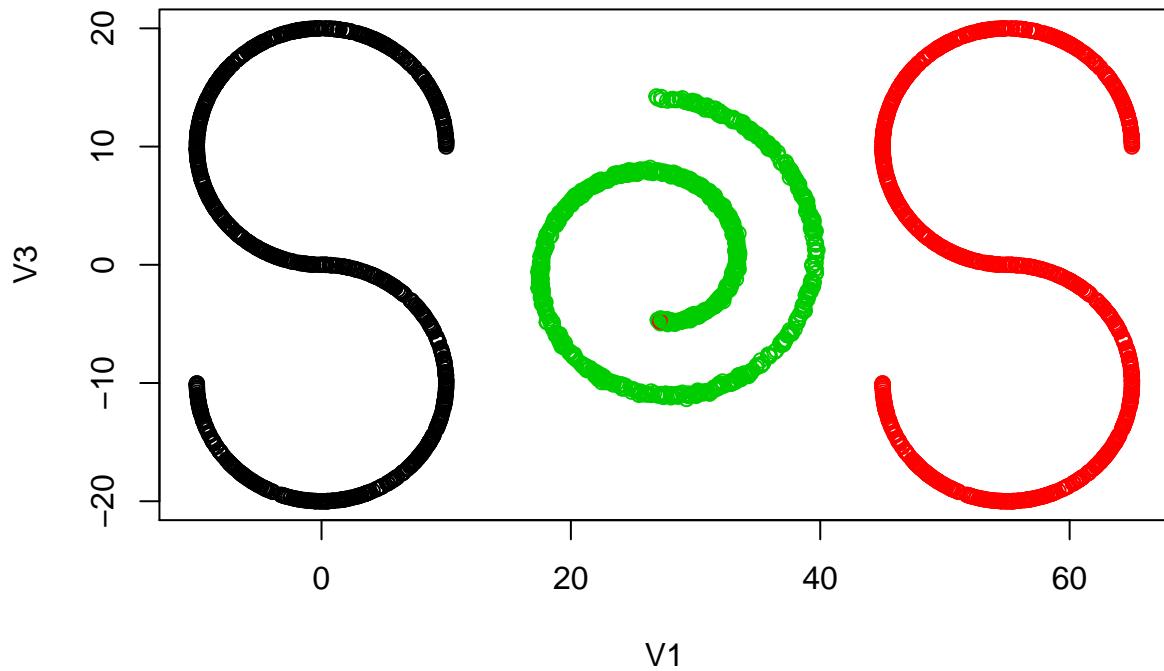
Vemos que las columnas relevantes son V_1 y V_3 , ya que con ellas se ve la figura formada por los datos (SOS). También vemos que los datos se encuentran separados en tres clusters similares a los de los archivos anteriores, por lo que podemos modificar las clases y reducirlas a 3.

```
> helpFile = helpFile[c("V1", "V3")]
> head(helpFile)
```

```
##          V1         V3
## 1  5.047159 18.632855
## 2 -8.961992  5.563481
## 3  6.534056 -2.429920
## 4  0.345659 -0.005976
## 5  3.074971 19.515490
## 6  5.144209 18.575378
```

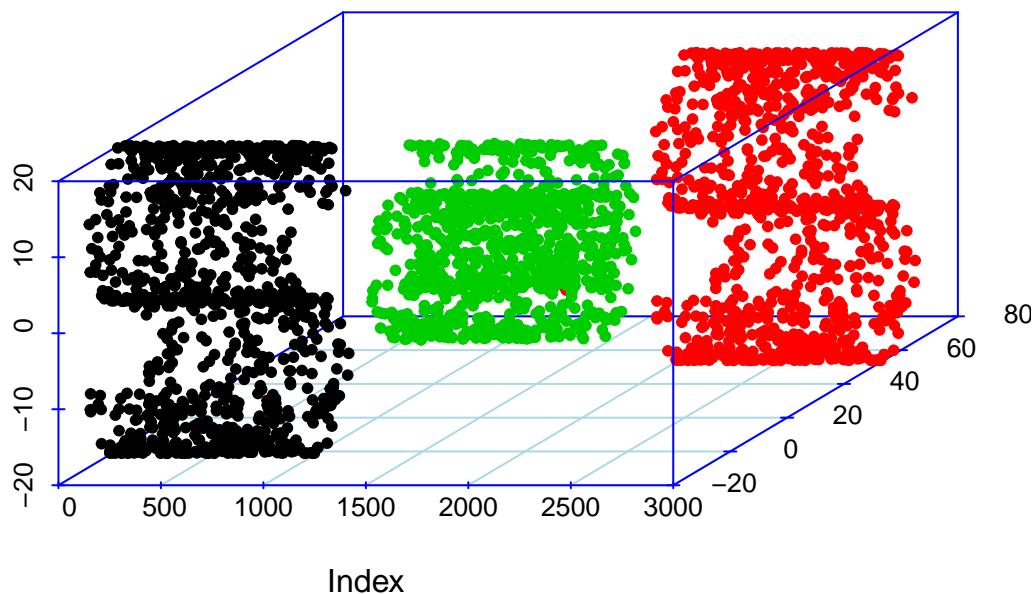
```
> helpClass[1:1000] = 1
> helpClass[2001:3000] = 2
>
> plot(helpFile$V1, helpFile$V3, col = helpClass, xlab = "V1", ylab = "V3", main = "Archivo help.csv")
```

Archivo help.csv



```
> scatterplot3d(helpFile, col.axis = "blue", col.grid = "lightblue", main = "help.csv en 3D",
+     pch = 20, color = helpClass)
```

help.csv en 3D



Debido a que el dataset posee tres clusters bien definidos y separados, usaremos k medias para intentar clasificar las instancias,

5. Usando *kmeans*

```
> helpFile.k.medias = kmeans(x = helpFile, centers = 3)
>
> # Graficamos los clusters obtenidos en kmeans
> plot(helpFile$V1, helpFile$V3, col = helpFile.k.medias$cluster, xlab = "V1",
+      ylab = "V3", main = "Archivo help.csv con kmedias")
>
> helpFile.k.medias$centers
```

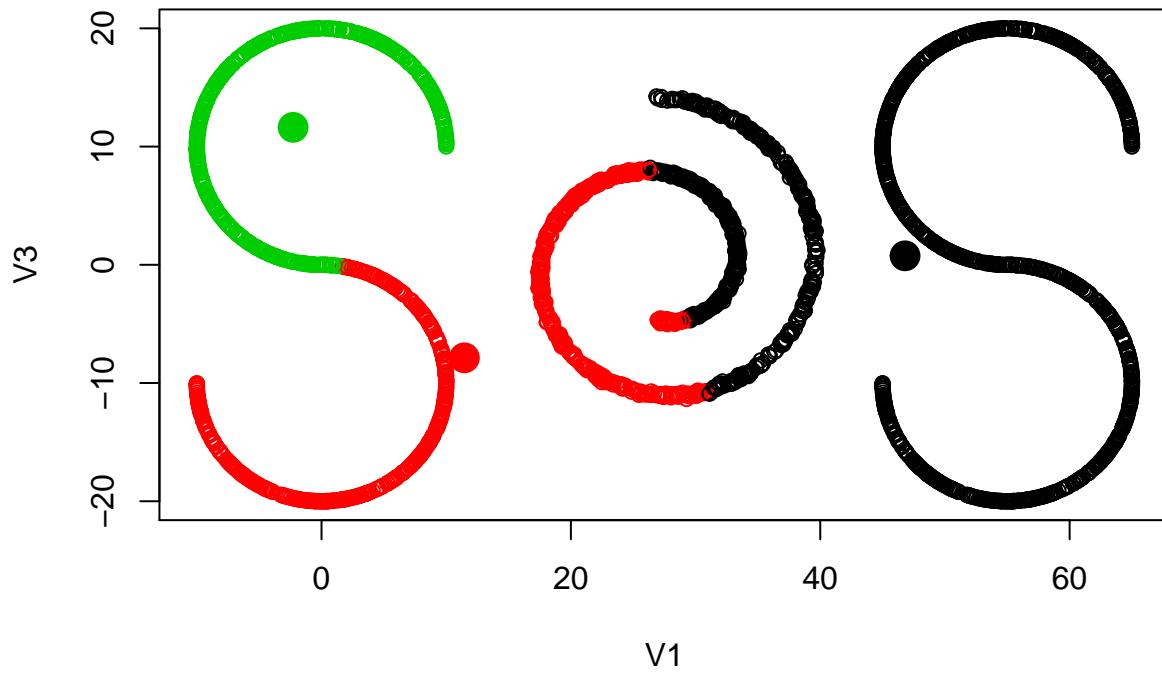


```
##           V1          V3
## 1 46.803382  0.7609089
## 2 11.457322 -7.8690992
## 3 -2.277815 11.6250901
```



```
> # Graficamos los centroides asociados a cada cluster Usamos un color
> # distinto para los centroides para verlos mejor
> points(x = helpFile.k.medias$centers[, c("V1", "V3")], col = 1:5, pch = 20,
+         cex = 3)
```

Archivo help.csv con kmedias



Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = helpFile.k.medias$cluster, y = helpClass, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
## |-----|  
## |           N |  
## |     N / Row Total |  
## |     N / Col Total |  
## |     N / Table Total |  
## |-----|  
##  
##  
## Total Observations in Table:  3000  
##  
##  
##  
##          | helpClass  
## helpFile.k.medias$cluster |    1 |    2 |    3 | Row Total |  
## -----|-----|-----|-----|-----|  
##  
##       1 |    0 | 1000 |   594 | 1594 |  
## | 0.000 | 0.627 | 0.373 | 0.531 |  
## | 0.000 | 0.999 | 0.595 | |  
## | 0.000 | 0.333 | 0.198 | |  
## -----|-----|-----|-----|-----|
```

```

##          2 |      478 |       1 |      405 |     884 |
##           | 0.541 | 0.001 | 0.458 | 0.295 |
##           | 0.478 | 0.001 | 0.405 |      |
##           | 0.159 | 0.000 | 0.135 |      |
## -----
##          3 |      522 |       0 |       0 |     522 |
##           | 1.000 | 0.000 | 0.000 | 0.174 |
##           | 0.522 | 0.000 | 0.000 |      |
##           | 0.174 | 0.000 | 0.000 |      |
## -----
##    Column Total | 1000 | 1001 | 999 | 3000 |
##           | 0.333 | 0.334 | 0.333 |      |
## -----
##
```

Detallando la matriz podemos observar que las etiquetas asignadas por k medias no corresponden a las originales.

Cambiamos esto y analizamos los resultados,

```

> helpClusters = helpFile.k.medias$cluster
>
> helpClusters[helpClusters == 1] = 4
> helpClusters[helpClusters == 2] = 5
> helpClusters[helpClusters == 3] = 6
>
> helpClusters[helpClusters == 4] = 2
> helpClusters[helpClusters == 5] = 3
> helpClusters[helpClusters == 6] = 1
>
> CrossTable(x = helpClusters, y = helpClass, prop.chisq = FALSE)

```

```

##
##
##   Cell Contents
## |-----|
## |           N |
## |           N / Row Total |
## |           N / Col Total |
## |           N / Table Total |
## |-----|
## 
## 
## Total Observations in Table: 3000
## 
## 
##          | helpClass
## helpClusters |      1 |      2 |      3 | Row Total |
## -----|-----|-----|-----|-----|
##      1 |      522 |       0 |       0 |     522 |
##           | 1.000 | 0.000 | 0.000 | 0.174 |
##           | 0.522 | 0.000 | 0.000 |      |
##           | 0.174 | 0.000 | 0.000 |      |
##
```

```

## -----|-----|-----|-----|-----|
##   2 |     0 |    1000 |     594 |    1594 |
##     | 0.000 |  0.627 |  0.373 |  0.531 |
##     | 0.000 |  0.999 |  0.595 |      |
##     | 0.000 |  0.333 |  0.198 |      |
## -----|-----|-----|-----|-----|
##   3 |    478 |      1 |    405 |    884 |
##     | 0.541 |  0.001 |  0.458 |  0.295 |
##     | 0.478 |  0.001 |  0.405 |      |
##     | 0.159 |  0.000 |  0.135 |      |
## -----|-----|-----|-----|-----|
## Column Total |    1000 |    1001 |    999 |    3000 |
##               |  0.333 |  0.334 |  0.333 |      |
## -----|-----|-----|-----|-----|
##
```

Con k medias solo hay error en una instancia, por lo tanto este método funciona casi perfectamente para este tipo de datasets.

Preguntas

- **Cuántos clusters ve en el dataset help?** Se ven 3 clusters bien definidos (S O S), dos S's y un espiral.
- **Qué pasa al aplicar la regla de asignación de clases en este dataset?** Ambas S's se dividen en dos clases (1 y 2), mientras que el espiral se corresponde con la clase 3. Por lo tanto, la regla no asigna correctamente las clases.
- **Qué solución daría para asignar de manera correcta los valores de las clases y pueda analizar el desempeño del algoritmo de clustering de manera correcta?** Determinamos en qué parte del dataset se encuentran las instancias asociadas a cada una de las S's y asignamos una clase distinta a cada una.

good_luck.csv

1. Cargamos el dataset *good_luck.csv*, asumiendo que se hizo **setwd(directorio)**:

```

> good_luckFile = read.csv(file = "./good_luck.csv", header = FALSE, sep = ",")
> head(good_luckFile)

```

```

##          V1         V2         V3         V4         V5         V6         V7
## 1 -0.262989 -0.868793  0.133290  2.745286 -0.047937  1.357079 -0.499947
## 2  0.114041 -1.274678  1.518768  2.227560  0.679981 -0.257101  1.847423
## 3  0.974706 -0.314599 -0.731224  1.742879  0.361086  1.872469  0.874509
## 4  0.175839 -1.120091  0.522660  0.461655  1.264471  0.390200  0.714736
## 5 -0.238445 -0.656013 -0.663740  0.879592 -0.176941  0.915897 -0.327490
## 6 -0.162517  1.680123  1.188765  0.864624 -0.393824  1.630972  0.614608
##          V8         V9         V10        V11
## 1 -1.874985 -0.395397  1.563203     1
## 2  0.586754  1.978936 -0.392775     1
## 3 -0.297938 -0.906587 -1.081991     1

```

```

## 4 -0.905856 0.456372 -0.697988 0
## 5 1.034972 0.732777 -0.257709 0
## 6 0.603824 0.294797 -0.649450 0

```

2. Separamos la última columna del dataset. La cual, hace referencia a la clase de cada instancia,

```

> good_luckClass = good_luckFile$V11
> good_luckFile = good_luckFile[c("V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8",
+   "V9", "V10")]
> head(good_luckFile)

```

```

##          V1          V2          V3          V4          V5          V6          V7
## 1 -0.262989 -0.868793  0.133290  2.745286 -0.047937  1.357079 -0.499947
## 2  0.114041 -1.274678  1.518768  2.227560  0.679981 -0.257101  1.847423
## 3  0.974706 -0.314599 -0.731224  1.742879  0.361086  1.872469  0.874509
## 4  0.175839 -1.120091  0.522660  0.461655  1.264471  0.390200  0.714736
## 5 -0.238445 -0.656013 -0.663740  0.879592 -0.176941  0.915897 -0.327490
## 6 -0.162517  1.680123  1.188765  0.864624 -0.393824  1.630972  0.614608
##          V8          V9          V10
## 1 -1.874985 -0.395397  1.563203
## 2  0.586754  1.978936 -0.392775
## 3 -0.297938 -0.906587 -1.081991
## 4 -0.905856  0.456372 -0.697988
## 5  1.034972  0.732777 -0.257709
## 6  0.603824  0.294797 -0.649450

```

```
> summary(good_luckClass)
```

```

##    Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 0.000 0.000 0.000 0.487 1.000 1.000

```

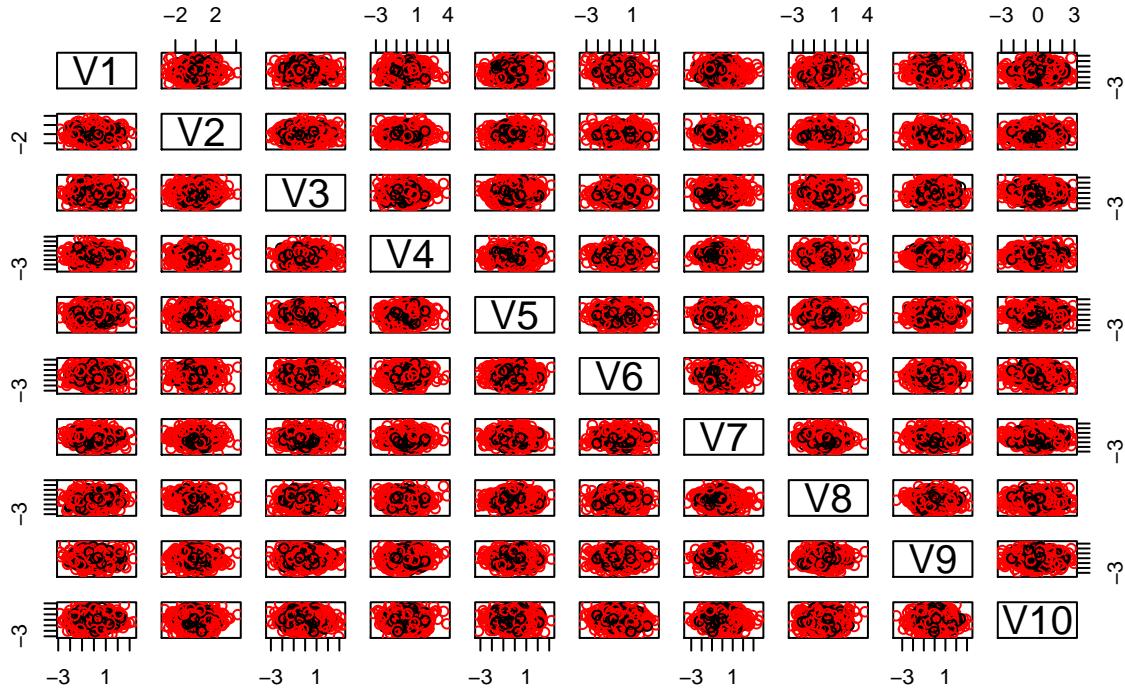
3. Ahora, hacemos un *plot* para verificar como se comportan los datos de *good_luck.csv*,

```

> # Sumamos uno a good_luck Class porque la clase cero es el color blanco
> plot(good_luckFile, col = good_luckClass + 1, main = "Archivo good_luck.csv")

```

Archivo good_luck.csv

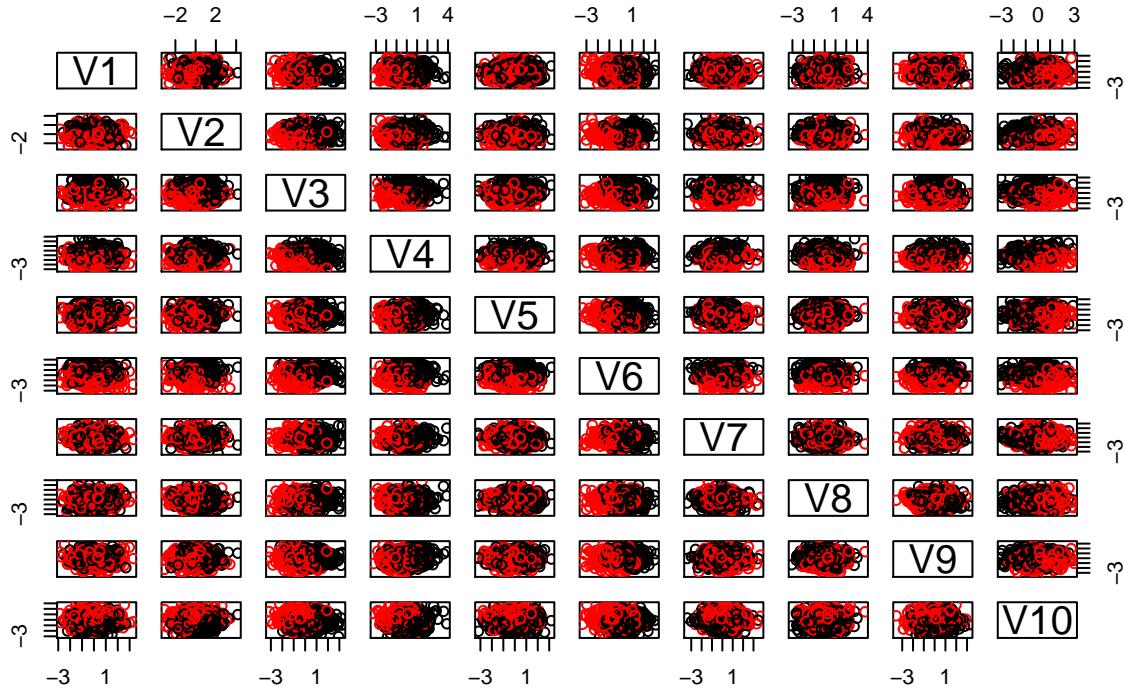


Vemos que los datos se separan en 2 clusters, pero al ser un dataset que posee 10 dimensiones, es imposible ver la forma exacta que poseen los datos. Parece ser algo similar a una esfera.

4. Usaremos k medias para tratar de predecir las clases,

```
> good_luckFile.k.medias = kmeans(x = good_luckFile, centers = 2)
>
> # Graficamos los clusters obtenidos en kmeans
> plot(good_luckFile, col = good_luckFile.k.medias$cluster, main = "Archivo good_luck.csv con kmedias")
```

Archivo good_luck.csv con kmedias



```
> good_luckFile.k.medias$centers
```

```
##          V1          V2          V3          V4          V5          V6
## 1 -0.020794627  0.2931667  0.5323961  0.3253657  0.1097405  0.4833605
## 2 -0.009065275 -0.2112859 -0.5082534 -0.4307308 -0.1388211 -0.4778889
##          V7          V8          V9          V10
## 1 -0.012287293 -0.06763631  0.1334016 -0.2708847
## 2  0.007844215  0.05218061 -0.1452935  0.2403082
```

Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = good_luckFile.k.medias$cluster, y = good_luckClass + 1, prop.chisq = FALSE)
```

```
##
##          Cell Contents
##          |-----|
##          |           N |
##          |           N / Row Total |
##          |           N / Col Total |
##          |           N / Table Total |
##          |-----|
##
```

```

## Total Observations in Table: 1000
##
##
##                               | good_luckClass + 1
## good_luckFile.k.medias$cluster |      1 |      2 | Row Total |
## -----
##           1 |     242 |    256 |      498 |
##           | 0.486 | 0.514 | 0.498 |
##           | 0.472 | 0.526 |      |
##           | 0.242 | 0.256 |      |
## -----
##           2 |     271 |    231 |      502 |
##           | 0.540 | 0.460 | 0.502 |
##           | 0.528 | 0.474 |      |
##           | 0.271 | 0.231 |      |
## -----
## Column Total |     513 |    487 |     1000 |
##           | 0.513 | 0.487 |      |
## -----
##
```

Se predicen de forma correcta 524 de 1000 (0.524) instancias, por lo tanto hubo error en 476 de 1000 (0.476).

Veamos que ocurre si usamos *hclust complete*

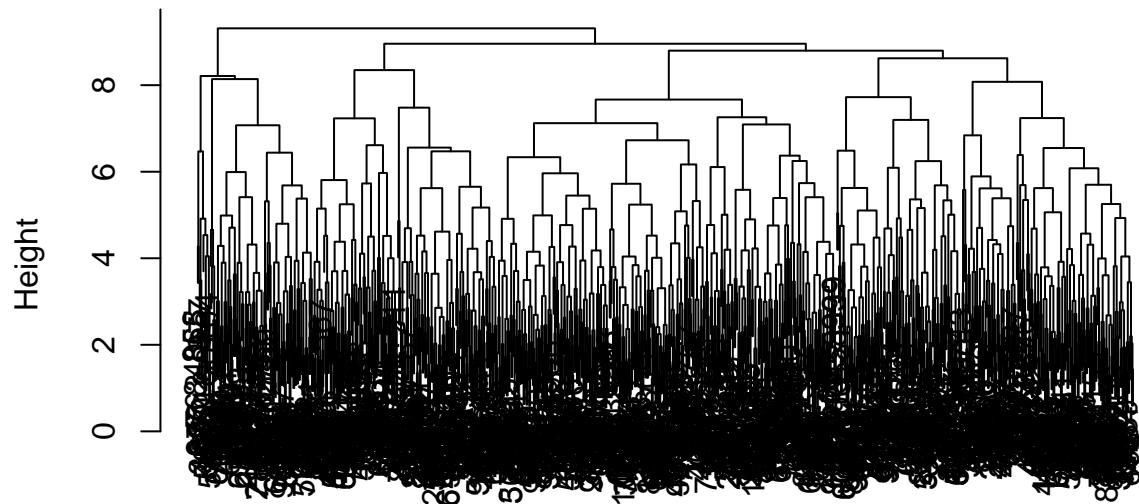
5. Usando *hclust complete*

```

> # Dataframe a Matrix
> good_luckFileMatrix = as.matrix(good_luckFile)
>
> # Matriz de distancia con norma 2 method = 'euclidean', 'maximum',
> # 'manhattan', 'canberra', 'binary' or 'minkowski'
> distancia = dist(good_luckFileMatrix, method = "euclidean")
>
> # Método por defecto es complete link
> #'single', 'complete'
> metodo = "complete"
>
> cluster = hclust(distancia, method = metodo)
>
> plot(cluster, main = "Cluster jerarquico de good_luck.csv")

```

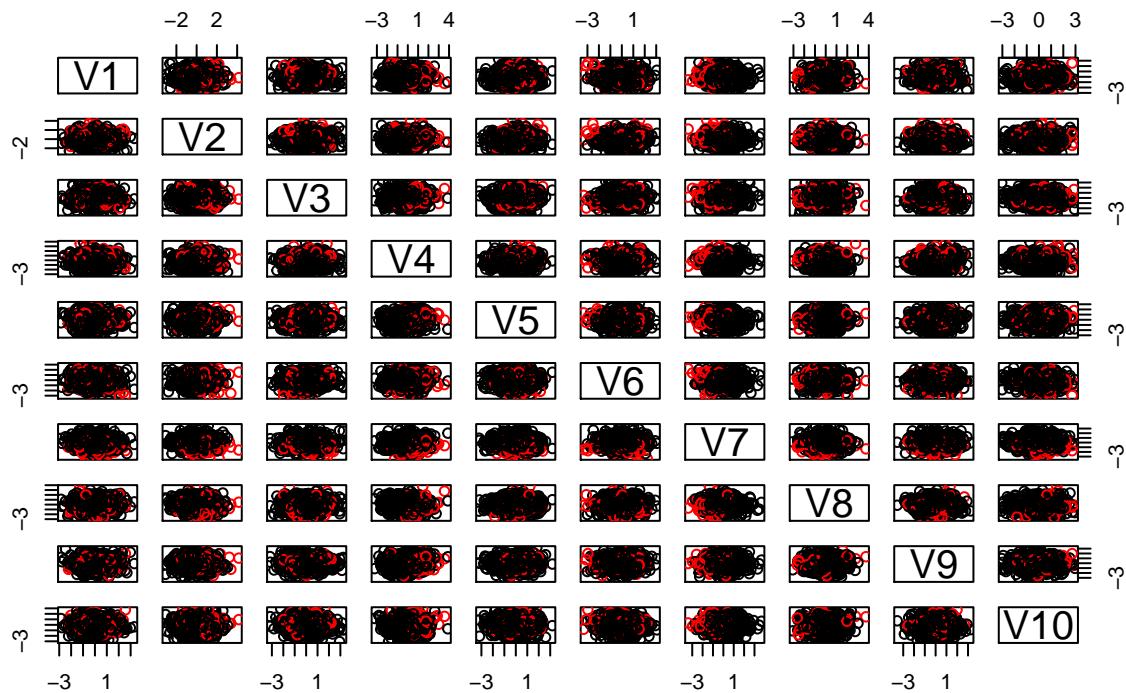
Cluster jerarquico de good_luck.csv



distancia
hclust (*, "complete")

```
> # Observamos dos clases en el analisis exploratorio
> nclases = 2
>
> # Cortamos
> corte = cutree(cluster, k = nclases)
>
> # Graficamos el dataset con el color de las clases obtenidas con hclust
> plot(good_luckFile, col = corte, main = "Archivo good_luck.csv hclust complete")
```

Archivo good_luck.csv hclust complete



Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = corte, y = good_luckClass + 1, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
## |-----|  
## |           N |  
## |           N / Row Total |  
## |           N / Col Total |  
## |           N / Table Total |  
## |-----|  
##  
##  
## Total Observations in Table:  1000  
##  
##  
##          | good_luckClass + 1  
## corte |       1 |       2 | Row Total |  
## -----|-----|-----|-----|  
##     1 |    467 |    409 |    876 |  
##     | 0.533 | 0.467 | 0.876 |  
##     | 0.910 | 0.840 |      |  
##     | 0.467 | 0.409 |      |  
## -----|-----|-----|-----|
```

```

##      2 |      46 |      78 |     124 |
##      | 0.371 | 0.629 | 0.124 |
##      | 0.090 | 0.160 |      |
##      | 0.046 | 0.078 |      |
## -----|-----|-----|-----|
## Column Total | 513 | 487 | 1000 |
##      | 0.513 | 0.487 |      |
## -----|-----|-----|-----|
##  

##
```

Se predicen de forma correcta 545 de 1000 (0.545) instancias, por lo tanto hubo error en 455 de 1000 (0.455).

Veamos que ocurre si usamos *hclust single*

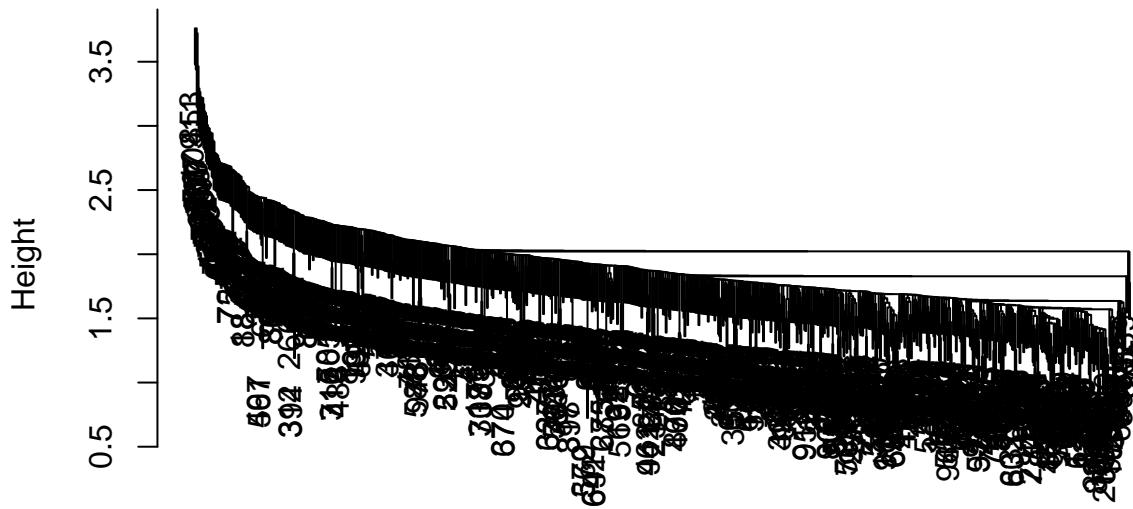
6. Usando *hclust single*

```

> # Dataframe a Matrix
> good_luckFileMatrix = as.matrix(good_luckFile)
>
> # Matriz de distancia con norma 2 method = 'euclidean', 'maximum',
> # 'manhattan', 'canberra', 'binary' or 'minkowski'
> distancia = dist(good_luckFileMatrix, method = "euclidean")
>
> # Método por defecto es complete link
> #'single', 'complete'
> metodo = "single"
>
> cluster = hclust(distancia, method = metodo)
>
> plot(cluster, main = "Cluster jerarquico de good_luck.csv")

```

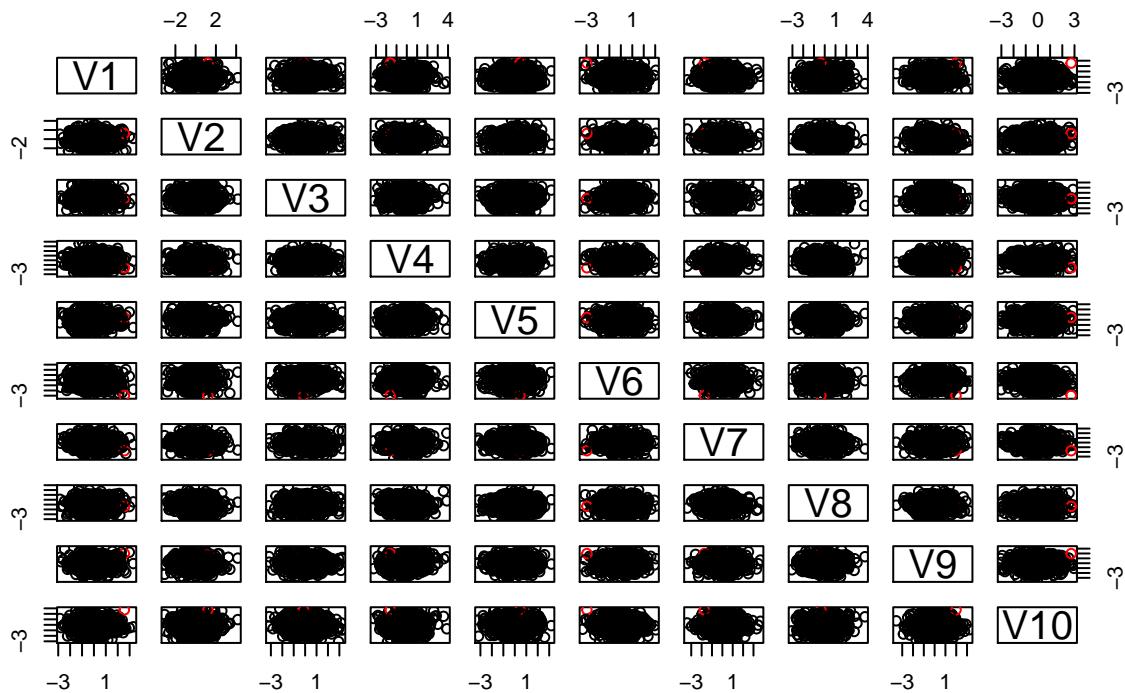
Cluster jerarquico de good_luck.csv



distancia
hclust (*, "single")

```
> # Observamos dos clases en el analisis exploratorio
> nclases = 2
>
> # Cortamos
> corte = cutree(cluster, k = nclases)
>
> # Graficamos el dataset con el color de las clases obtenidas con hclust
> plot(good_luckFile, col = corte, main = "Archivo good_luck.csv hclust single")
```

Archivo good_luck.csv hclust single



Verificamos la calidad del modelo con una matriz de confusión

```
> CrossTable(x = corte, y = good_luckClass + 1, prop.chisq = FALSE)
```

```
##  
##  
##      Cell Contents  
## |-----|  
## |           N |  
## |           N / Row Total |  
## |           N / Col Total |  
## |           N / Table Total |  
## |-----|  
##  
##  
## Total Observations in Table:  1000  
##  
##  
##  
##           | good_luckClass + 1  
## corte |     1 |     2 | Row Total |  
## -----|-----|-----|-----|  
## 1 | 513 | 486 | 999 |  
## | 0.514 | 0.486 | 0.999 |  
## | 1.000 | 0.998 | |  
## | 0.513 | 0.486 | |  
## -----|-----|-----|-----|
```

```

##      2 |      0 |      1 |      1 |
##      | 0.000 | 1.000 | 0.001 |
##      | 0.000 | 0.002 |      |
##      | 0.000 | 0.001 |      |
## -----|-----|-----|-----|
## Column Total | 513 | 487 | 1000 |
##      | 0.513 | 0.487 |      |
## -----|-----|-----|-----|
##
```

Se predicen de forma correcta 514 de 1000 (0.514) instancias, por lo tanto hubo error en 486 de 1000 (0.486).

7. Conclusión:

Ninguno de los métodos usados funciona de forma correcta debido a que el set de datos es de 10 dimensiones, pero de tener que elegir alguno, podría ser *kmeans* o *hclust complete link* porque con estos se comete la menor cantidad de errores.

Definición de la función *codoJambu()*

El codo de jambu es un método que nos permite predecir el k o la cantidad de clases usadas en k medias.

```

> codo_jambu = function(x, n) {
+   InerciaIC = rep(0, n)
+   for (k in 1:n) {
+     K = kmeans(x, k, nstart = n)
+     InerciaIC[k] = K$tot.withinss
+   }
+   plot(InerciaIC, col = "skyblue", type = "b", main = "Codo de Jambu con n iteraciones")
+ }
```

guess.csv

1. Cargamos el dataset *guess.csv*, asumiendo que se hizo **setwd(directorio)**:

```

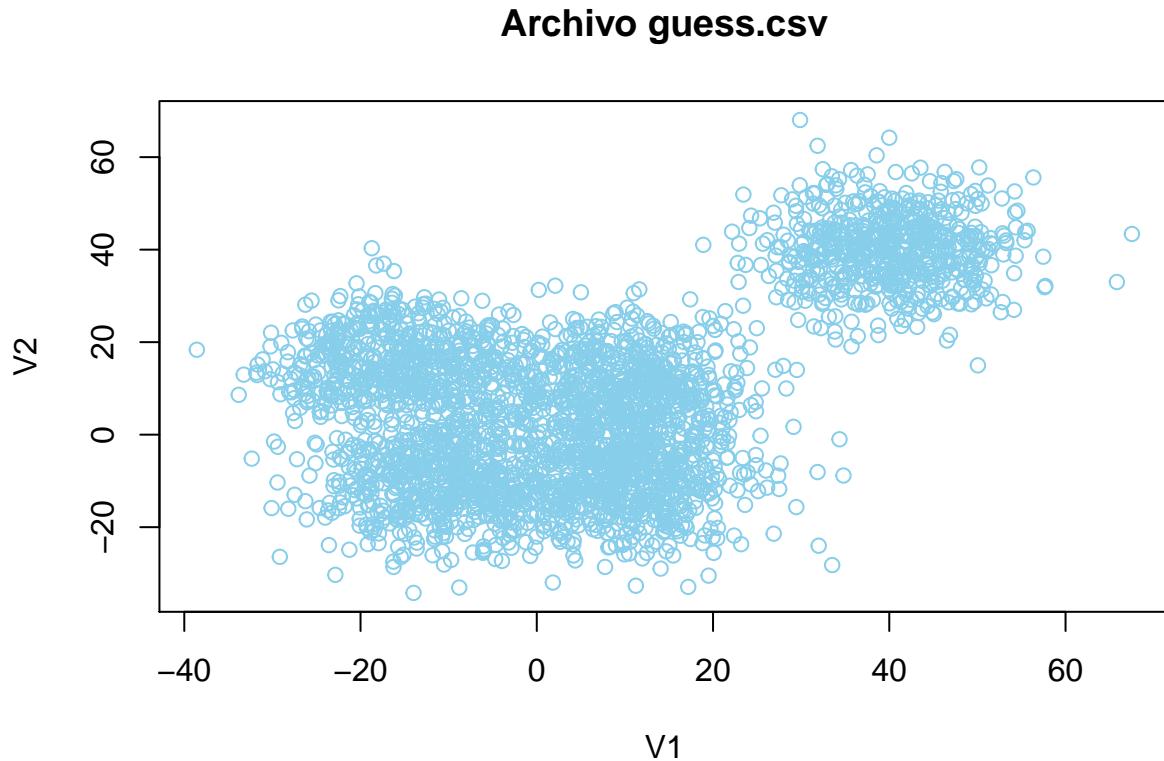
> guessFile = read.csv(file = "./guess.csv", header = FALSE, sep = ", ")
> head(guessFile)
```

```

##          V1        V2
## 1 -22.856666 -30.302967
## 2  17.637588  8.613766
## 3 -24.046328 19.647426
## 4 -25.918317 12.200390
## 5   5.908121 -18.359366
## 6   8.395719 -15.811708
```

2. Ahora, hacemos un *plot* para verificar como se comportan los datos de *guess.csv*,

```
> plot(guessFile$V1, guessFile$V2, col = "skyblue", xlab = "V1", ylab = "V2",
+       main = "Archivo guess.csv")
```

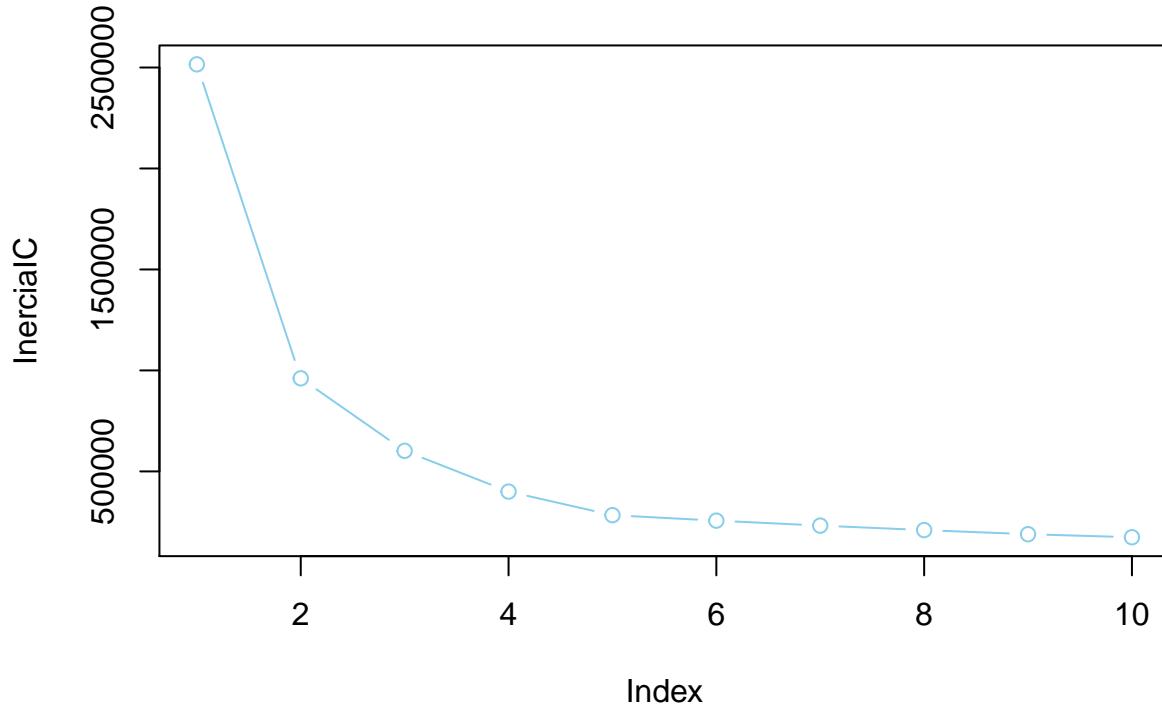


No se observa bien la cantidad de clusters que posee el dataset, parecen ser 3 ó 4. Usemos la función *codo_jambu()* para determinar el k.

3. Llamamos a *codo_jambu()*

```
> codo_jambu(guessFile, n = 10)
```

Codo de Jambu con n iteraciones

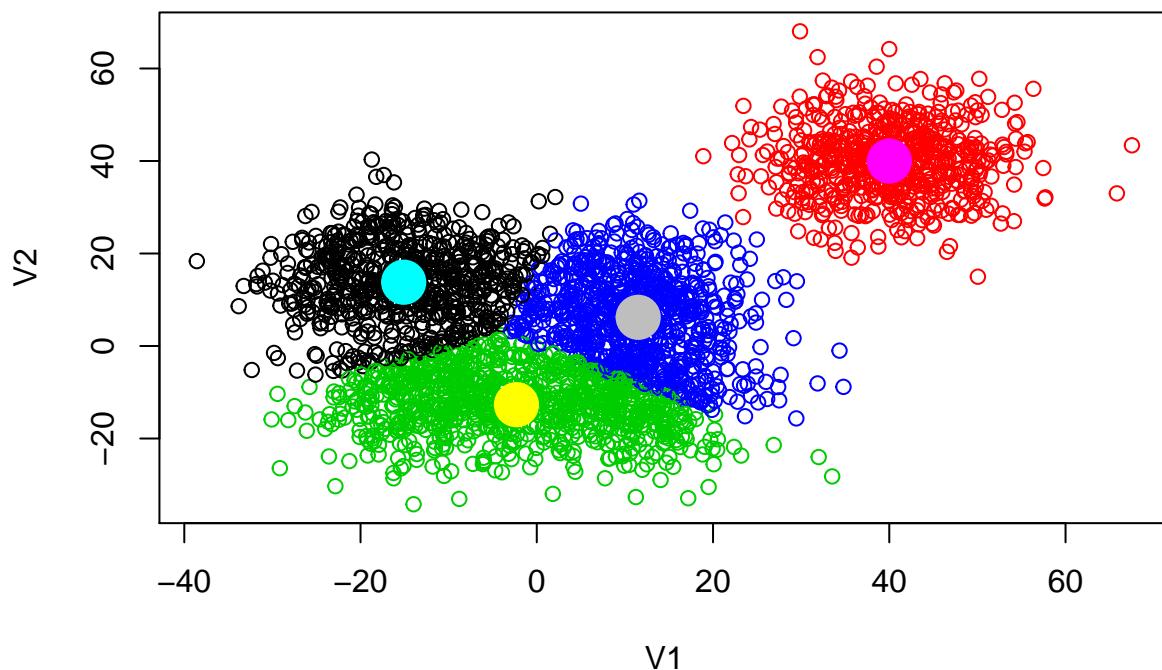


Veamos que a partir de **k=4** no varia mucho la curva del codo de jambu y este es el punto que usaremos para aplicar kmedias.

4. Usamos la función kmeans y verificamos que tan bien predice las clases del dataset con **k=4**, porque este fue el número hallado con el codo de Jambu,

```
> guessFile.k.medias = kmeans(x = guessFile, centers = 4)
>
> # Graficamos los clusters obtenidos en kmeans
> plot(guessFile$V1, guessFile$V2, col = guessFile.k.medias$cluster, xlab = "V1",
+      ylab = "V2", main = "Archivo guess.csv")
>
> # Graficamos los centroides asociados a cada cluster Usamos un color
> # distinto para los centroides para verlos mejor
> points(x = guessFile.k.medias$centers[, c("V1", "V2")], col = 5:8, pch = 19,
+         cex = 3)
```

Archivo guess.csv



Efectivamente, con $k=4$ el algoritmo separa de forma correcta los clusters del dataset *guess.csv*

Definición de la función *kmeans()*

```
> # c -> centroides x -> dataframe with instances
>
> K_Means = function(centers, x) {
+   xClasses = vector(length = length(x[, 1])) #vector con las clases
+   for (j in 1:length(x[, 1])) {
+     # iterar en instancias
+     distVector = c(1:length(centers[, 1])) #Vector donde se almacenan las
+     # distancias de una instancia a cada centro recorremos todos los centros
+     for (i in 1:length(centers[, 1])) {
+       df = x[j, ] #instancia actual
+       df[2, ] = centers[i, ] #anexamos la instancia al dataframe
+       distVector[i] = dist(df, method = "euclidean")[1] #norma 2 entre la instancia y el centro
+     }
+
+     xClasses[j] = which.min(distVector) #Seleccionamos la menor distancia y
+     # la posicion de la misma es la clase de la instancia j
+   }
+
+   xClasses
+ }
```

a_big.csv

1. Cargamos el dataset *a_big.csv*, asumiendo que se hizo `setwd(directorio)`:

```
> a_bigFile = read.csv(file = "./a_big.csv", header = FALSE, sep = ",")  
> head(a_bigFile)
```

```
##          V1          V2  V3  
## 1  8.694977 -7.254259  2  
## 2 11.537866  9.091760  0  
## 3 -6.672032 -8.277564  1  
## 4 11.003768  9.930918  0  
## 5  8.762679  1.518045  0  
## 6 10.325578 14.677636  0
```

2. Separamos la última columna del dataset. La cual, hace referencia a la clase de cada instancia,

```
> a_bigClass = a_bigFile$V3  
> a_bigFile = a_bigFile[c("V1", "V2")]  
> head(a_bigFile)
```

```
##          V1          V2  
## 1  8.694977 -7.254259  
## 2 11.537866  9.091760  
## 3 -6.672032 -8.277564  
## 4 11.003768  9.930918  
## 5  8.762679  1.518045  
## 6 10.325578 14.677636
```

3. Ahora, hacemos un *plot* para verificar como se comportan los datos de *a_big.csv*,

```
> summary(a_bigClass)  
  
##    Min. 1st Qu. Median     Mean 3rd Qu.    Max.  
##      0      0      1      1      2      2  
  
> # Sumamos uno a aClass porque la clase cero es el color blanco  
> # plot(a_bigFile$V1, a_bigFile$V2, col = a_bigClass + 1, xlab = 'V1', ylab =  
> # 'V2', main = 'Archivo a_big.csv')  
>  
> length(a_bigFile$V1)  
  
## [1] 300000
```

Vemos que los datos se separan en 3 clusters circulares que se encuentran unidos y es un poco lento el proceso porque el dataset posee 300000 instancias.

4. Usamos la función `kmeans` y verificamos que tan bien predice las clases del dataset con **k=3**, porque este fue el número de clusters hallados en el análisis exploratorio,

```

> a_bigFile.k.medias = kmeans(x = a_bigFile, centers = 3)
>
> # Graficamos los clusters obtenidos en kmeans plot(a_bigFile$V1,
> # a_bigFile$V2, col = a_bigFile.k.medias$cluster, xlab = 'V1', ylab = 'V2',
> # main = 'Archivo a_big.csv kmeans')
>
> # Graficamos los centroides asociados a cada cluster Usamos un color
> # distinto para los centroides para verlos mejor points(x =
> # a_bigFile.k.medias$centers[, c('V1', 'V2')], col = 4:6, pch = 19, cex = 3)

```

5. Verificamos la calidad del modelo con una matriz de confusión

```

> CrossTable(x = a_bigFile.k.medias$cluster, y = a_bigClass, prop.chisq = FALSE)

## 
## 
##   Cell Contents
##   |-----|
##   |           N |
##   |           N / Row Total |
##   |           N / Col Total |
##   |           N / Table Total |
##   |-----|
## 
## 
## Total Observations in Table:  300000
## 
## 
##          | a_bigClass
## a_bigFile.k.medias$cluster |      0 |      1 |      2 | Row Total |
## -----|-----|-----|-----|-----|
##       1 | 99405 |     14 |    666 | 100085 |
##             | 0.993 | 0.000 | 0.007 | 0.334 |
##             | 0.994 | 0.000 | 0.007 | |
##             | 0.331 | 0.000 | 0.002 | |
## -----|-----|-----|-----|-----|
##       2 |     11 | 99401 |    589 | 100001 |
##             | 0.000 | 0.994 | 0.006 | 0.333 |
##             | 0.000 | 0.994 | 0.006 | |
##             | 0.000 | 0.331 | 0.002 | |
## -----|-----|-----|-----|-----|
##       3 |     584 |     585 | 98745 | 99914 |
##             | 0.006 | 0.006 | 0.988 | 0.333 |
##             | 0.006 | 0.006 | 0.987 | |
##             | 0.002 | 0.002 | 0.329 | |
## -----|-----|-----|-----|-----|
## Column Total | 100000 | 100000 | 100000 | 300000 |
##             | 0.333 | 0.333 | 0.333 | |
## -----|-----|-----|-----|-----|
## 
## 
```

Vemos que hay error en 2449 (0.008), lo cual es no mucho comparado con el gran tamaño del dataset.

6. Usando la función *kmeans* el proceso tarda un poco, veamos que pasa si usamos los centroides de *a.csv*, dado que este archivo es un subconjunto de *a_big.csv*. Además usaremos la función *K_Means* implementada en la sección anterior.

```
> # a_bigFile.kmeans = K_Means(x = a_bigFile, centers = aCentroids)
>
> # Graficamos los clusters obtenidos en kmeans plot(a_bigFile$V1,
> # a_bigFile$V2, col = a_bigFile.kmeans, xlab = 'V1', ylab = 'V2', main =
> # 'Archivo a_big.csv kmeans propio')
>
> # Graficamos los centroides asociados a cada cluster Usamos un color
> # distinto para los centroides para verlos mejor points(x = aCentroids, col
> # = 4:6, pch = 19, cex = 3)
```

7. Verificamos la calidad del modelo con una matriz de confusión

```
> # CrossTable(x=a_bigFile.kmeans, y=a_bigClass, prop.chisq = FALSE)
```