

Aprendizaje No Supervisado

Deyban Pérez

Abril 3, 2016

Abstract

En **Aprendizaje no Supervisado** no se cuenta con la sección de clases reales de los elementos del conjunto de entrenamiento, cosa que complica un poco las cosas y hace esencial un buen uso de análisis exploratorio de los datos para poder determinar los algoritmos y parámetros adecuados para lograr un buen modelo.

En esta ocasión vamos a utilizar cuatro técnicas de aprendizaje no supervisado:

- K-Medias
- H-Clust Single
- H-Clust Complete
- H-Clust Average

Para implementar modelos a ocho (8) conjuntos de datos y compararemos cuál se ajusta mejor al problema y el motivo del mismo.

Estructura

Vamos a dividir el problema en tres (3) partes:

- **Análisis con clases enteras:** donde la columna clase corresponde a valores enteros que no necesitan transformaciones para definir las clases.
- **Análisis con clases reales:** donde la columna clase corresponde a valores reales que ameritan de una transformación para definir las clases.
- **Análisis sin columna de clase:** donde no hay columna de clase y se amerita aplicar en su totalidad la técnica de aprendizaje no supervisado.

Dependencias

Utilizaremos la biblioteca “scatterplot3d” de R que nos permite realizar gráficos en 3D de una manera rápida y sencilla que nos ayudará a hacer un buen análisis exploratorio más adelante.

Para instalarar dicho paquete, ejecutar el siguiente comando en la terminal de **Ubuntu**:

- sudo apt-get install r-cran-rgl

Una vez que se haya instalado la biblioteca, debemos cargarla, haciendo uso del siguiente comando:

```
library(scatterplot3d)
```

Implementación

Sólo utilizaremos la biblioteca **scatterplot3d** como biblioteca externa, todas las demás ya vienen por defecto en **R**, ahora vamos a definir algunas funciones que nos ayudarán a facilitar nuestro trabajo.

Funciones

Pre-procesamiento de datos

El rango de las clases en **a.csv**, **a_big.csv**, **moon.csv** y **good_luck.csv** está en el rango de valores [0,1], es conveniente hacer que el rango empiece en uno (1) y no en cero (0) debido a que los modelos empiezan sus clasificaciones a partir del número uno (1).

```
pre_processing = function(df)
{
  df$V3 = as.numeric(df$V3)
  df$V3 = df$V3 - min(df$V3) + 1

  return(df)
}
```

Pre-Procesamiento de datos especial para **good_luck.csv**

El conjunto de datos **good_luck.csv** Tiene la columna de clases en la posición once (11), por ello se creó una función especial para este caso.

```
pre_processing_especial = function(df)
{
  df$V11 = as.numeric(df$V11)
  df$V11 = df$V11 - min(df$V11) + 1

  return(df)
}
```

Extensión del rango de colores

Por defecto el rango de colores en R es de ocho (8) y son reutilizados, en conjuntos de datos como **h.csv** y **s.csv** veremos cómo hay más de ocho (8) clases y es oportuno poder visualizarlas gráficamente con colores diferentes.

```
get_colors = function(number, df)
{
  set.seed(22)
  index = sample(1:502,number, replace = F)
  colors = colors(distinct = T)[index]
  return(colors[as.integer(df$V5-min(df$V5)+1)])
}
```

Evaluación de K-Medias para conjuntos 2D

Esta función genera el modelo de **K-Medias** y grafica cómo queda el conjunto de datos una vez que se realizó el modelo.

```
eval_kmeans = function(df, cstart, cfinish, k, dataname)
{
  model_kmeans = kmeans(x = df[, cstart:cfinish], centers = k)
  plot(df[, cstart:cfinish], col = model_kmeans$cluster,
       main = paste(c("Data set", dataname), collapse = " "), sub = "K-means algorithm",
       xlab = "Feature 1", ylab = "Feature 2")
  points(model_kmeans$centers[,c("V1", "V2")],
         col = 6:8,
         pch = 19,
         cex = 2)

  return(model_kmeans)
}
```

Evaluación de K-Medias para conjuntos 2D especial para good_luck.csv

Esta función genera el modelo de **K-Medias** y grafica cómo queda el conjunto de datos una vez que se realizó el modelo, sin embargo el gráfico es sólo de la matriz de dispersión entre todas las componentes.

```
eval_kmeans_especial = function(df, cstart, cfinish, k, dataname)
{
  model_kmeans = kmeans(x = df[, cstart:cfinish], centers = k)
  plot(df[,1:10], col = df$V11,
       main = paste(c("Matriz de Dispersion",dataname), collapse = " "),
       sub = "K-Means Algorithm")
  return(model_kmeans)
}
```

Evaluación de K-Medias para conjuntos 3D

Esta función genera el modelo de **K-Medias** y grafica cómo queda el conjunto de datos una vez que se realizó el modelo en 3D.

```
eval_kmeans_3D = function(df, cstart, cfinish, k, dataname)
{
  model_kmeans = kmeans(x = df[, cstart:cfinish], centers = k)
  scatterplot3d(x = df$V1, y = df$V2, z = df$V3, type = "p" ,color = model_kmeans$cluster,
                xlab = "Feature 1", ylab = "Feature 2", zlab = "Feature 3",
                main = paste(c("Data Set",dataname), collapse = " "))
  return(model_kmeans)
}
```

Evaluación de H-Clust para conjuntos 2D

Función que genera el modelo H-Clust con el tipo de método pasado como parámetro y grafica el resultado de aplicar el modelo.

```

eval_hclust = function(distance, mode, centroids, input, dataname)
{
  model = hclust(distance, method = mode)
  model_cut= cutree(model, k = centroids)
  plot(x = input[,1], y = input[,2],
    main = paste(c("Data set", dataname), collapse = " "),
    sub = paste(c(mode,"H-Clust Algorithm"), collapse = " "),
    xlab = "Feature 1", ylab = "Feature 2",
    col = model_cut)
  return(model_cut)
}

```

Evaluación de H-Clust para conjuntos 2D especial para good_luck.csv

Función que genera el modelo H-Clust con el tipo de método pasado como parámetro y grafica el resultado de aplicar el modelo, pero imprime la matriz de dispersión entre las componentes.

```

eval_hclust_especial = function(distance, mode, centroids, input, dataname,cfinish)
{
  model = hclust(distance, method = mode)
  model_cut= cutree(model, k = centroids)
  input = as.data.frame(input)
  plot(input[,1:cfinish],
    main = paste(c("Data set", dataname), collapse = " "),
    sub = paste(c(mode,"H-Clust Algorithm"), collapse = " "),
    col = model_cut)
  return(model_cut)
}

```

Evaluación de H-Clust para conjuntos 3D

Función que genera el modelo H-Clust con el tipo de método pasado como parámetro y grafica el resultado de aplicar el modelo en 3D.

```

eval_hclust_3D = function(distance, mode, centroids, input, dataname)
{
  model = hclust(distance, method = mode)
  model_cut= cutree(model, k = centroids)
  scatterplot3d(x = input[,1], y = input[,2], z = input[,3], type = "p",
    main = paste(c("Data set", dataname), collapse = " "),
    sub = paste(c(mode,"H-Clust Algorithm"), collapse = " "),
    xlab = "Feature 1", ylab = "Feature 2", zlab = "Feature 3",
    color = model_cut)
  return(model_cut)
}

```

Generación de la matriz de confusión

Función que genera la matriz de confusión entre las clases reales y las predichas por un modelo.

```

confusion_matrix = function(class, cluster)
{
  return(table(True = class, Predicted = cluster))
}

```

Comprobación de vector vacío

Función que retorna **TRUE** si un vector está lleno de ceros y **FALSE** en caso contrario.

```

zeros_colum = function(column)
{
  for (i in 1:length(column))
  {
    if(column[i] > 0)
      return(FALSE)
  }
  return(TRUE)
}

```

Suma de diagonal

Función que retorna la suma de la diagonal de una matriz de confusión.

```

sum_diagonal = function(matrix)
{
  returnValue = 0;

  for (i in 1:ncol(matrix))
  {
    returnValue = returnValue + matrix[i,i]
  }

  return(returnValue)
}

```

Tasa de acierto del modelo

Función que retorna la tasa de acierto y de fallo de un modelo.

```

confusion_matrix_evaluation = function(confusionMatrix, numberOfRows)
{
  hits = sum_diagonal(confusionMatrix)
  hits
  hitRate = hits / numberOfRows * 100
  missRate = 100 - hitRate
  write(sprintf("Tasa de acierto: %f", hitRate), stdout())
  write(sprintf("Tasa de fallo: %f", missRate), stdout())
}

```

Ordenar matriz de confusión

En aprendizaje no supervisado la matriz de confusión puede salir desordenada, por esto, es importante ordenarla para poder hacer análisis de los modelos.

```
order_confusion_matrix = function(confusionMatrix)
{
  auxiliar = matrix(0, nrow = nrow(confusionMatrix), ncol = ncol(confusionMatrix))

  for (i in 1:ncol(confusionMatrix))
  {
    positions = which(confusionMatrix == max(confusionMatrix), arr.ind = T)

    #If the column is empty
    if(zeros_colum(auxiliar[, positions[1,1]]))
    {
      auxiliar[, positions[1,1]] = confusionMatrix[,positions[1,2]]
      confusionMatrix[,positions[1,2]] = vector(mode = "numeric",
                                                length = nrow(confusionMatrix))
    }
    else
    {
      for (j in 1:ncol(auxiliar))
      {
        if(zeros_colum(auxiliar[,j]))
          break
      }

      if(auxiliar[positions[1,1], positions[1,1]] >= confusionMatrix[positions[1,1],
                                                                positions[1,2]])
      {
        auxiliar[, j] = confusionMatrix[,positions[1,2]]
        confusionMatrix[,positions[1,2]] = vector(mode = "numeric",
                                                length = nrow(confusionMatrix))
      }
      else
      {
        auxiliar[, j] = auxiliar[, positions[1,1]]
        auxiliar[, positions[1,1]] = confusionMatrix[,positions[1,2]]
        confusionMatrix[,positions[1,2]] = vector(mode = "numeric",
                                                length = nrow(confusionMatrix))
      }
    }
  }

  rownames(auxiliar) <- rownames(auxiliar, do.NULL = FALSE, prefix = "TC")
  colnames(auxiliar) <- colnames(auxiliar, do.NULL = FALSE, prefix = "PC")

  return(auxiliar)
}
```

Tasa de acierto del modelo para mi implementación de K-Means

Función que retorna exclusivamente la tasa de acierto de un modelo.

```

confusion_matrix_evaluation_deyban = function(confusionMatrix, numberOfRows)
{
  hits = sum_diagonal(confusionMatrix)
  hits
  hitRate = hits / numberOfRows * 100
  return(hitRate)
}

```

Función que retorna la distancia euclídea entre dos puntos

```

euc.dist = function(x1, y1, x2, y2)
{
  return( sqrt( ((x1 - x2) ^ 2) + ((y1 - y2) ^ 2)) )
}

```

Implementación de K-Means

```

K_Means_Deyban = function(df, k, c = NULL)
{
  #Selecting random centroids
  centroids = matrix(0L, nrow = k, ncol = 2)

  # If You dont have centroids, choose random centroids
  if(is.null(c))
  {
    for (i in 1:nrow(centroids))
    {
      centroids[i,1] = runif(1, min(df$V1), max(df$V1))
      centroids[i,2] = runif(1, min(df$V2), max(df$V2))
    }
  }else #Take centroids from parameters
  centroids = c

  #Initializing clusters vector
  clusters = vector(mode = "numeric", length = nrow(df))
  #Initializing Distance Matrix
  distance_matrix = matrix(0L, nrow = nrow(df), ncol = nrow(centroids))

  #Initializing the number of elements by cluster
  c1 = 0
  c2 = 0
  c3 = 0

  # Beginning the algorithm
  for(n in 1:50)
  {
    #Calculating the distance between points and centroids
    for (i in 1:nrow(centroids))
      distance_matrix[,i] = euc.dist(centroids[i,1],
                                     centroids[i,2], df[,1], df[,2])
  }
}

```

```

#Assigning elements to a cluster
for (i in 1:length(clusters))
  clusters[i] = which.min(distance_matrix[i,])

#Finding new centroids
for (i in 1:nrow(centroids))
{
  centroids[i,1] = (1/length(clusters[clusters == i])) *
    sum(df[clusters[clusters == i],1])

  centroids[i,2] = (1/length(clusters[clusters == i])) *
    sum(df[clusters[clusters == i],2])
}

#Evaluating the change about clusters between iteration
if(n > 1)
{
  c1.temp = length(clusters[clusters == 1])
  c2.temp = length(clusters[clusters == 2])
  c3.temp = length(clusters[clusters == 3])

  # If the elements between iterations dont have changes
  if((c1 == c1.temp) && (c2 == c2.temp) && (c3 == c3.temp))
  {
    #return
    my_list = list(centroids, clusters)
    return(my_list)
  }
  else # Update values
  {
    c1 = c1.temp
    c2 = c2.temp
    c3 = c3.temp
  }
}
#Return
my_list = list(centroids, clusters)
return(my_list)
}

```

Análisis con clases enteras

Consta de tres conjuntos de datos:

- a.csv
- a_big.csv
- moon.csv
- good_luck.csv

Vamos a empezar a trabajar en ellos.

a.csv

Cargando el conjunto de datos

```
df_a = read.csv("../data/a.csv", header = F)
```

Pre-procesamiento

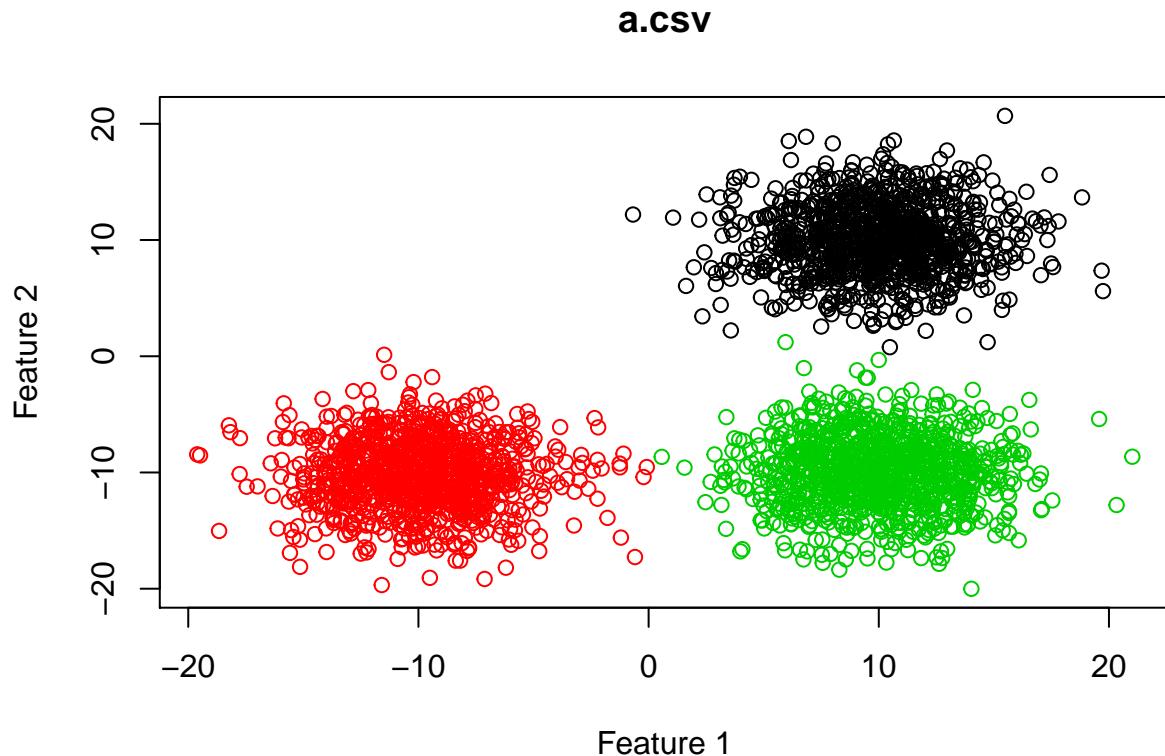
La columna de clases está en el rango [0,1] y la llevaremos al rango [1,2].

```
df_a = pre_processing(df_a)
```

Visualizando el conjunto de datos

Realizamos la gráfica para poder analizar al conjunto de datos a.csv.

```
plot(df_a[,1:2], col = df_a$V3,
      xlab = "Feature 1", ylab = "Feature 2",
      main = "a.csv")
```



En el gráfico podemos observar claramente la presencia de tres (3) clusters con una forma más o menos circular en cada uno de ellos.

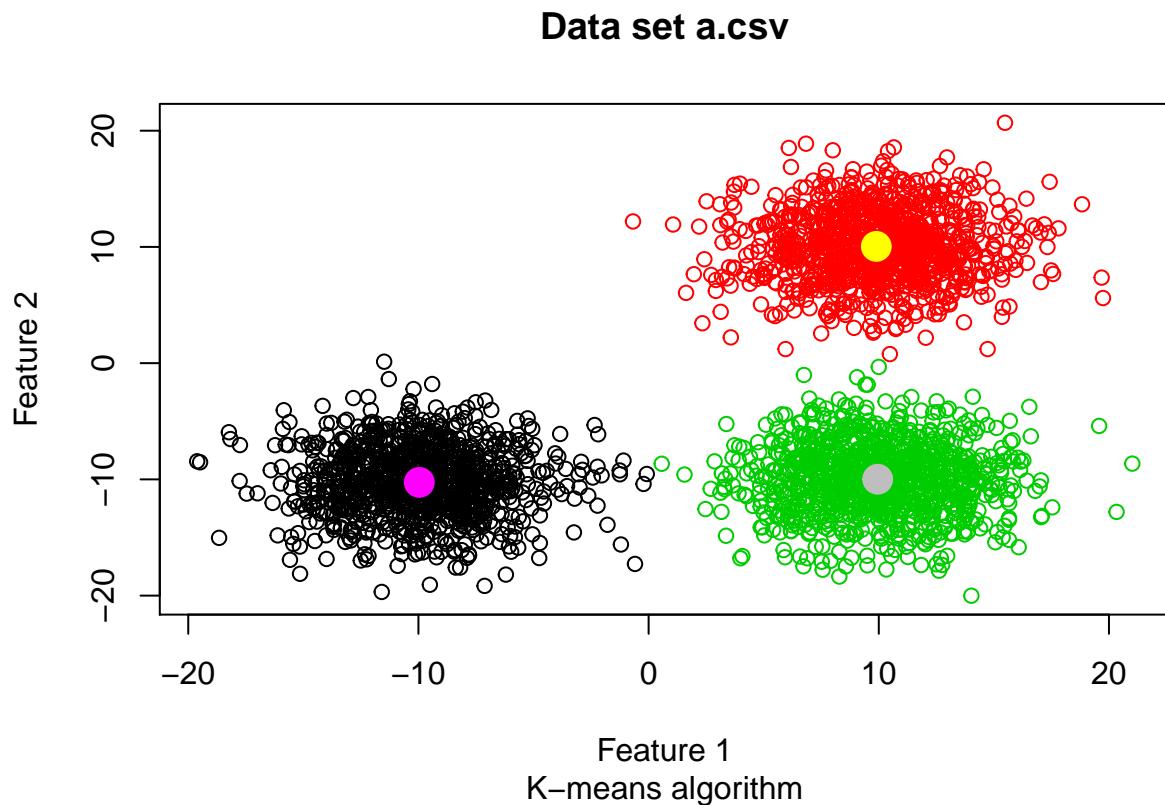
Aplicando los algoritmos

Vamos a probar los diferentes algoritmos de clusterización mencionados anteriormente para evaluar su desempeño.

K-Medias

Generando el modelo

```
model_kmeans_a = eval_kmeans(df = df_a, cstart = 1, cfinish = 2, k = 3, dataname = "a.csv")
```



Generando la matriz de confusión

```
table_model_kmeans_a = order_confusion_matrix(  
  confusion_matrix(df_a$V3, model_kmeans_a$cluster))  
table_model_kmeans_a
```

```
##      PC1  PC2 PC3  
## TC1 1000    0   0  
## TC2    0 1000   0  
## TC3    1    0 999
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_kmeans_a, nrow(df_a))
```

```
## Tasa de acierto: 99.966667  
## Tasa de fallo: 0.033333
```

Conclusión

K-Medias funciona muy bien para este conjunto de datos por la naturaleza del algoritmo de buscar figuras circulares o superficies cilíndricas, en este caso se puede apreciar una alta tasa de acierto.

H-Clust

Necesitamos calcular la **matriz de distancias** antes de aplicar el algoritmo.

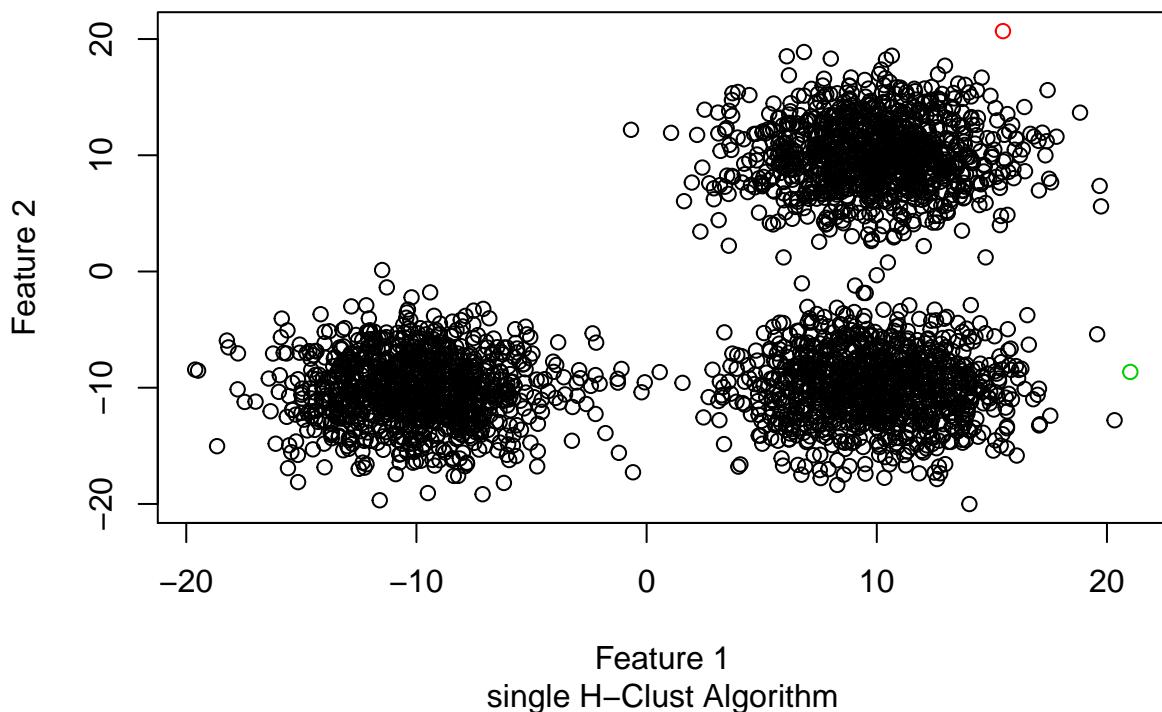
```
input_hierarchical_a = df_a
input_hierarchical_a$V3 = NULL
input_hierarchical_a = as.matrix(input_hierarchical_a)
hierarchical_distance_a = dist(input_hierarchical_a)
```

H-Clust Single

Generando el modelo

```
model_hierarchical_single_a = eval_hclust(
  distance = hierarchical_distance_a, mode = "single", centroids = 3,
  input = input_hierarchical_a, dataname = "a.csv")
```

Data set a.csv



Generando la matriz de confusión

```
table_model_hierarchical_single_a = order_confusion_matrix(  
  confusion_matrix(df_a$V3, model_hierarchical_single_a))  
table_model_hierarchical_single_a
```

```
##      PC1  PC2  PC3  
## TC1    1  999    0  
## TC2    0 1000    0  
## TC3    0   999    1
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_single_a, nrow(df_a))  
  
## Tasa de acierto: 33.400000  
## Tasa de fallo: 66.600000
```

Conclusión

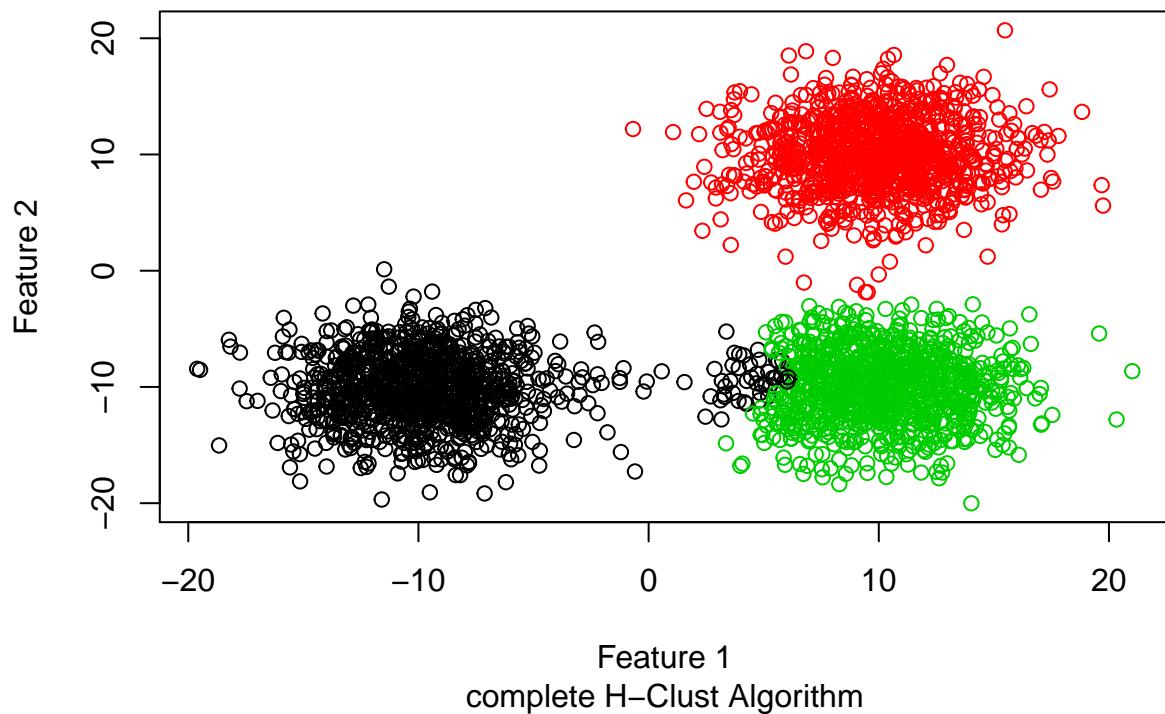
H-Clust single no muestra un gran rendimiento debido a que el algoritmo no está diseñado para buscar figuras circulares o cilíndricas, sino figuras no circulares.

H-Clust Complete

Generando el modelo

```
model_hierarchical_complete_a = eval_hclust(  
  distance = hierarchical_distance_a, mode = "complete",  
  centroids = 3, input = input_hierarchical_a, dataname = "a.csv")
```

Data set a.csv



Feature 1
complete H–Clust Algorithm

Generando la matriz de confusión

```
table_model_hierarchical_complete_a = order_confusion_matrix(  
  confusion_matrix(df_a$V3, model_hierarchical_complete_a))  
table_model_hierarchical_complete_a
```

```
##      PC1  PC2 PC3  
## TC1 1000    0   0  
## TC2    0 1000   0  
## TC3    6   47 947
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_complete_a, nrow(df_a))
```

```
## Tasa de acierto: 98.233333  
## Tasa de fallo: 1.766667
```

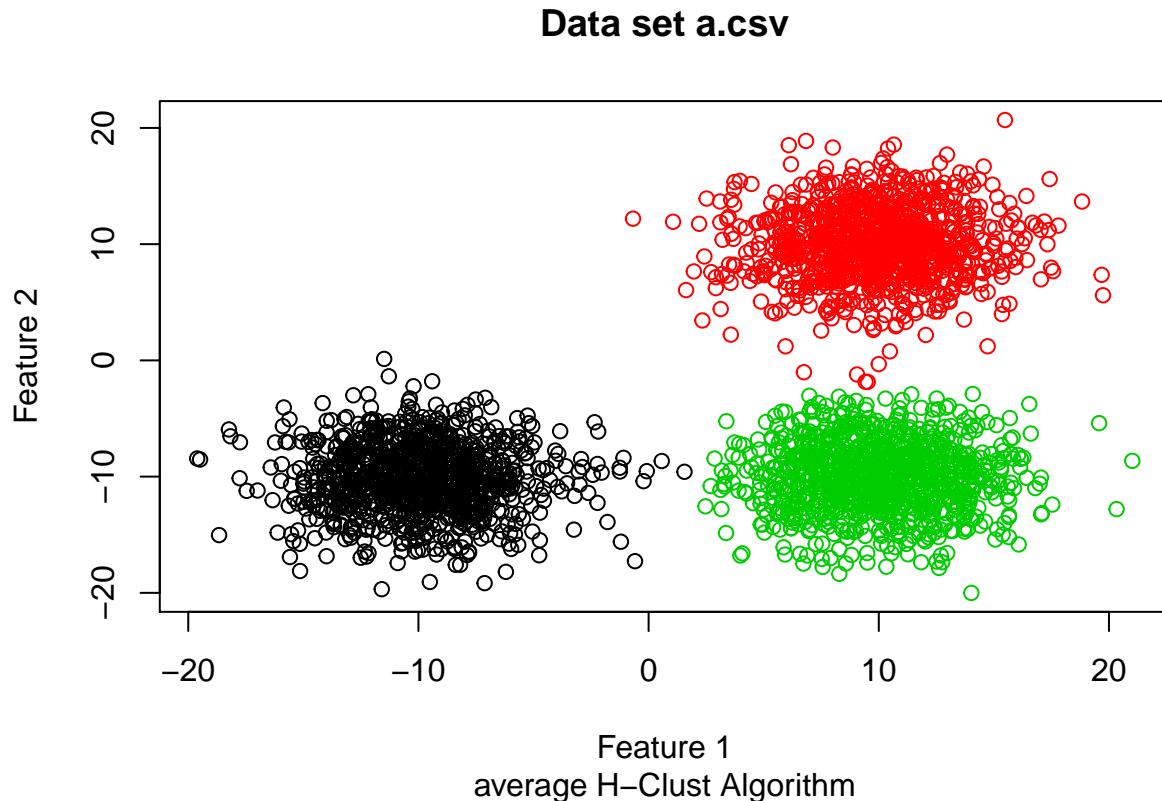
Conclusión

H-Clust complete se comporta parecido a K-Medias con respecto a que trata de buscar figuras circulares o cilíndricas, sin embargo en este caso no funcionó de la mejor manera, ya que confunde parte de los elementos de un cluster notablemente, disminuyendo así la tasa de aciertos.

H-Clust Average

Generando el modelo

```
model_hierarchical_average_a = eval_hclust(  
  distance = hierarchical_distance_a, mode = "average", centroids = 3,  
  input = input_hierarchical_a, dataname = "a.csv")
```



Generando la matriz de confusión

```
table_model_hierarchical_average_a = order_confusion_matrix(  
  confusion_matrix(df_a$V3, model_hierarchical_average_a))  
table_model_hierarchical_average_a
```

```
##      PC1  PC2  PC3  
## TC1 1000    0    0  
## TC2    0 1000    0  
## TC3    6    2 992
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_average_a, nrow(df_a))  
  
## Tasa de acierto: 99.733333  
## Tasa de fallo: 0.266667
```

Conclusión

H-clust average se comporta muy bien ya que está diseñado para buscar formas circulares, pero no completamente circulares sino un poco más dispersas, que es el caso de nuestro conjunto de datos.

Conclusión general

Por la estructura circular del conjunto de datos, todos los algoritmos menos **H-Clust Single** se comportan de una manera excelente, teniendo un promedio de 99% de acierto en eficacia.

a_big.csv

Cargando el conjunto de datos

```
df_a_big = read.csv("../data/a_big.csv", header = F)
```

Pre-procesamiento

La columna de clases está en el rango [0,1] y la llevaremos al rango [1,2].

```
df_a_big = pre_processing(df_a_big)
```

Visualizando el conjunto de datos

Primero vamos a ver la cantidad de elementos que tenemos en **a_big-csv**.

```
nrow(df_a_big)
```

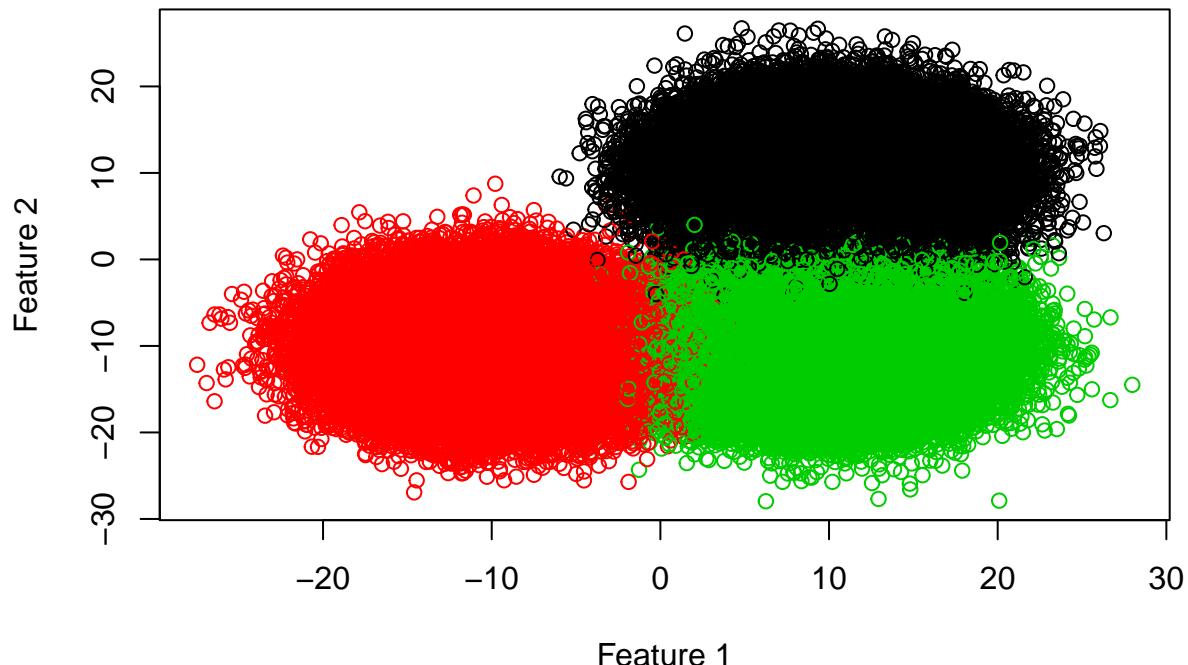
```
## [1] 300000
```

Vemos que tenemos **trescientas mil** (300000) ocurrencias que es un número bastante grande. Y debido a esto no podremos aplicar algoritmos **H-Clust** debido a que el tamaño de la matriz de distancias no cabe en memoria principal.

Veamos la gráfica para poder analizar al conjunto de datos **a_big.csv**

```
plot(df_a_big[,1:2], col = df_a_big$V3,
      xlab = "Feature 1", ylab = "Feature 2",
      main = "a_big.csv")
```

a_big.csv



En el gráfico podemos observar tres (3) clusters juntos con una forma más o menos circular, por lo tanto el algoritmo de **K-Medias** debería funcionar de buena manera.

Aplicando los algoritmos

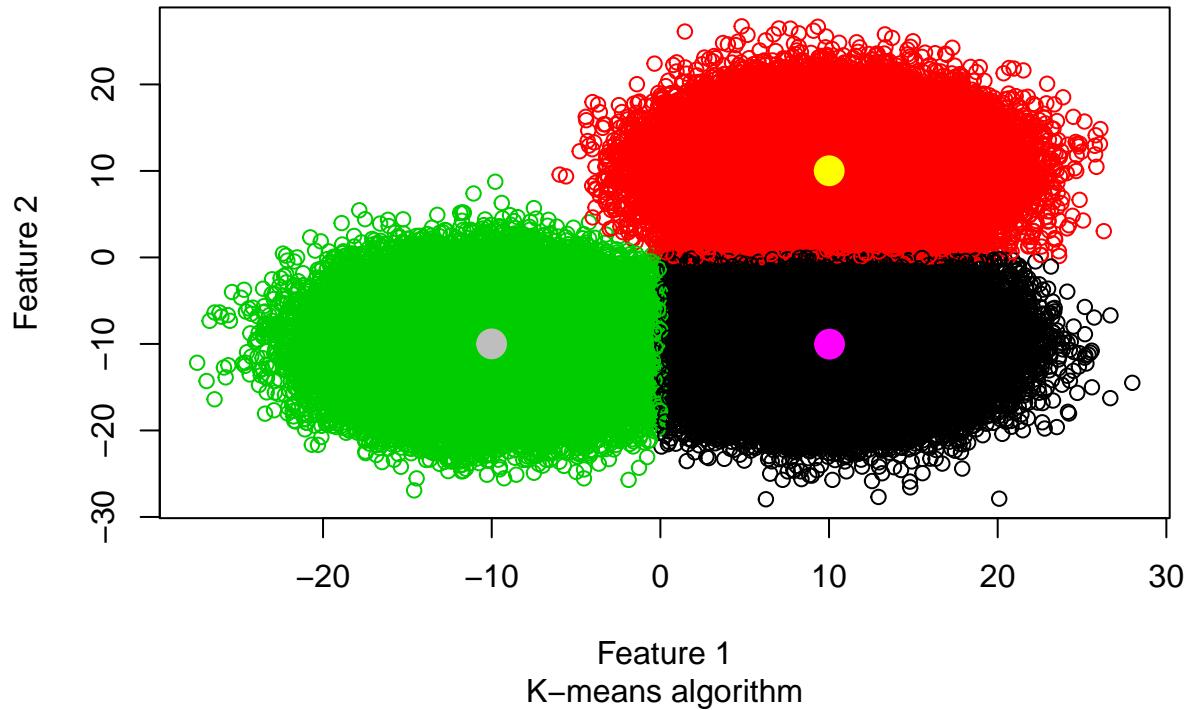
Cómo se mencionó más arriba, sólo podremos aplicar el algoritmo de **K-Medias** para clusterizar debido a que con **H-Clust** la matriz de distancias es demasiado grande y no cabe en memoria principal, comenzaremos primero con la versión del algoritmo propuesto por **R**.

K-Medias

Generando el modelo

```
model_kmeans_a_big = eval_kmeans(df = df_a_big, cstart = 1,  
                                  cfinish = 2, k = 3, dataname = "a_big.csv")
```

Data set a_big.csv



Generando la matriz de confusión

```
table_model_kmeans_a_big = order_confusion_matrix(  
  confusion_matrix(df_a_big$V3, model_kmeans_a_big$cluster))  
table_model_kmeans_a_big
```

```
##          PC1     PC2     PC3  
## TC1  99405      11    584  
## TC2      14  99401    585  
## TC3     666     589  98745
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_kmeans_a_big, nrow(df_a_big))
```

```
## Tasa de acierto: 99.183667  
## Tasa de fallo: 0.816333
```

Conclusión

K-Medias por su naturaleza de buscar formas circulares se ajusta de manera excelente a este problema, adicionalmente la implementación hace que el conjunto de datos gigante no sea un problema, pero es una variante especial llamada **Large k-Means** el algoritmo que se aplica por detrás de la llamada a la función.

**K-Means personalizado

Debido al gran tamaño de **a_big.csv** es conveniente hacer un muestreo reducido del conjunto total de los datos para correr el algoritmo y pre-seleccionar centroides que ayuden a la velocidad de convergencia cuando tratemos con el conjunto completo de los datos.

Sampling de los Datos

Calculando la probabilidad de cada elemento para ser seleccionado

```
prob_1 = 1/sum(df_a_big[, "V3"] == 1)
prob_2 = 1/sum(df_a_big[, "V3"] == 2)
prob_3 = 1/sum(df_a_big[, "V3"] == 3)
```

Reservando espacio para el vector de probabilidades

```
probabilities = vector(mode = "numeric", length = nrow(df_a_big))
```

Asignando las probabilidades a cada elemento

```
probabilities[df_a_big[, "V3"] == 1] = prob_1
probabilities[df_a_big[, "V3"] == 2] = prob_2
probabilities[df_a_big[, "V3"] == 3] = prob_3
```

Tomando la muestra del conjunto completo

```
set.seed(22)
best = 0

sub = sample(nrow(df_a_big), floor(nrow(df_a_big) * 0.1),
             prob = probabilities, replace = F)

subset <- df_a_big[sub, ]
```

**Precalculando los mejores centroides

```
for (i in 1:50)
{
  model_temporal = K_Means_Deyban(subset, k = 3)
  table_model_kmeans_deyban_a_big = order_confusion_matrix(
    confusion_matrix(subset$V3, model_temporal[[2]]))
  temporal = confusion_matrix_evaluation_deyban(
    table_model_kmeans_deyban_a_big, nrow(subset))

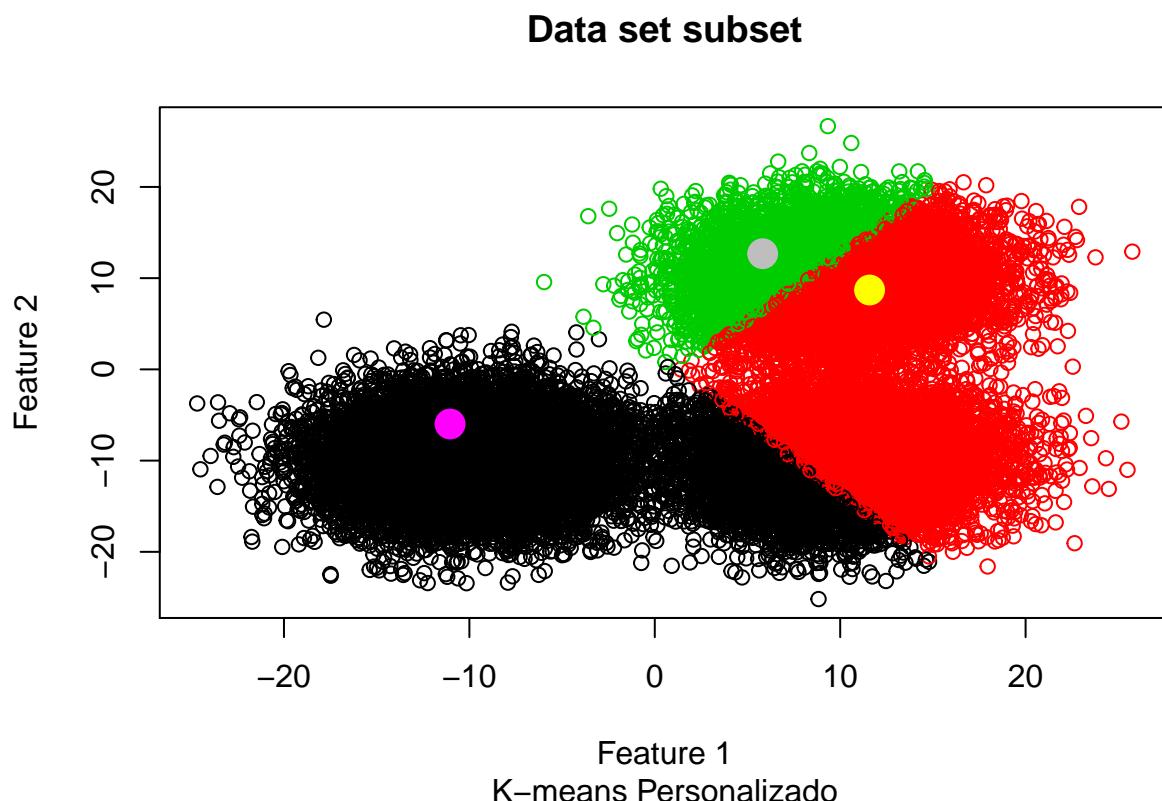
  if(temporal > best)
  {
    best_centroids = model_temporal[[1]]
    best_clusters = model_temporal[[2]]
    best = temporal
  }
}
```

}

Visualizando los centroides pre-calculados

```
plot(subset[, 1:2], col = best_clusters,
      main = paste(c("Data set", "subset"), collapse = " "),
      sub = "K-means Personalizado",
      xlab = "Feature 1", ylab = "Feature 2")

points(best_centroids[,1:2],
       col = 6:8,
       pch = 19,
       cex = 2)
```



Generando el modelo real

```
model big = K Means Deyban(df a big, 3, best centroids)
```

Calculando la matriz de confusión

```
table_model_kmeans_model_big = order_confusion_matrix(  
    confusion_matrix(df_a_big$V3, model_big[[2]]))  
table model kmeans model big
```

```

##          PC1     PC2     PC3
## TC1  98694      49   1257
## TC2       2 99739    259
## TC3     383  1327 98290

```

Calculando la tasa de acierto

```
confusion_matrix_evaluation(table_model_kmeans_model_big, nrow(df_a_big))
```

```

## Tasa de acierto: 98.907667
## Tasa de fallo: 1.092333

```

Visualizando la salida

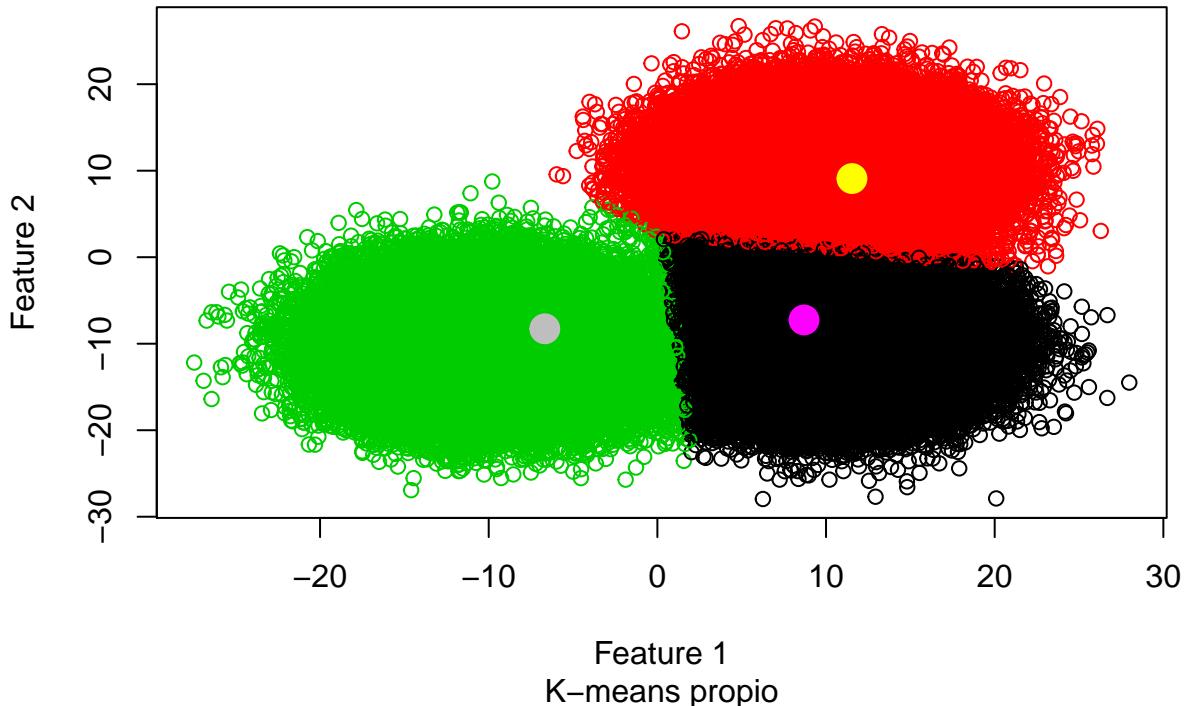
```

plot(df_a_big[, 1:2], col = model_big[[2]],
      main = paste(c("Data set", "a_big.csv"), collapse = " "), sub = "K-means propio",
      xlab = "Feature 1", ylab = "Feature 2")

points(model_big[[1]][,1:2],
      col = 6:8,
      pch = 19,
      cex = 2)

```

Data set a_big.csv



Conclusión

El algoritmo implementado se comporta de una manera bastante aceptable, es importante remarcar el hecho de que pre-calcular los centroides ayuda a que la convergencia sea mucho más rápida y posee unos valores parecidos a los centroides reales obtenidos por el algoritmo propuesto en la función **kmeans** de R.

moon.csv

Cargando el conjunto de datos

```
df_moon = read.csv("../data/moon.csv", header = F)
```

Pre-procesamiento

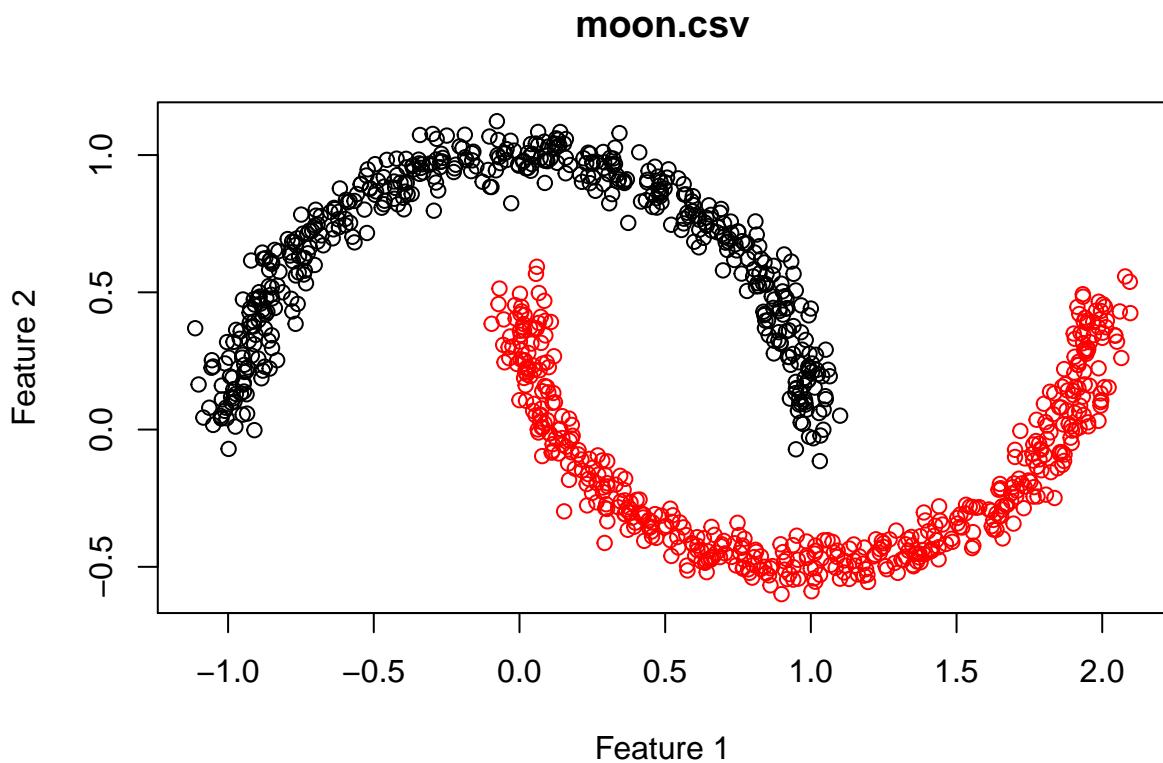
La columna de clases está en el rango [0,1] y la llevaremos al rango [1,2].

```
df_moon = pre_processing(df_moon)
```

Visualizando el conjunto de datos

Realizamos la gráfica para poder analizar al conjunto de datos **moon.csv**.

```
plot(df_moon[,1:2], col = df_moon$V3,
      xlab = "Feature 1", ylab = "Feature 2",
      main = "moon.csv")
```



En el gráfico podemos observar dos (2) clusters con forma de semi-luna, por la estructura, el algoritmo **H-Clust Single** debería ser el que mejor se comporte.

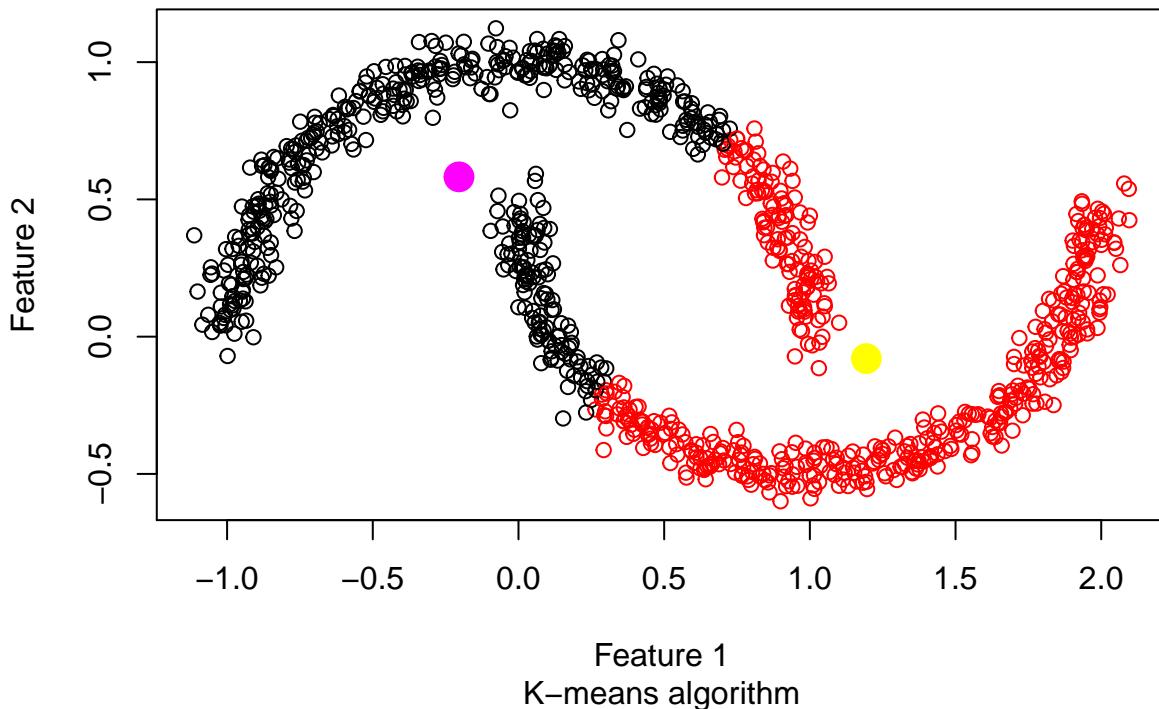
Aplicando los algoritmos

K-Medias

Generando el modelo

```
model_kmeans_moon = eval_kmeans(df = df_moon, cstart = 1,  
                                 cfinish = 2, k = 2, dataname = "moon.csv")
```

Data set moon.csv



Generando la matriz de confusión

```
table_model_kmeans_moon = order_confusion_matrix(  
    confusion_matrix(df_moon$V3, model_kmeans_moon$cluster))  
table_model_kmeans_moon
```

```
##      PC1 PC2  
## TC1 377 123  
## TC2 121 379
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_kmeans_moon, nrow(df_moon))
```

```
## Tasa de acierto: 75.600000  
## Tasa de fallo: 24.400000
```

Conclusión

El algoritmo K-Medias no tiene un buen desempeño por su naturaleza de buscar figuras circulares o superficies cilíndricas.

H-Clust

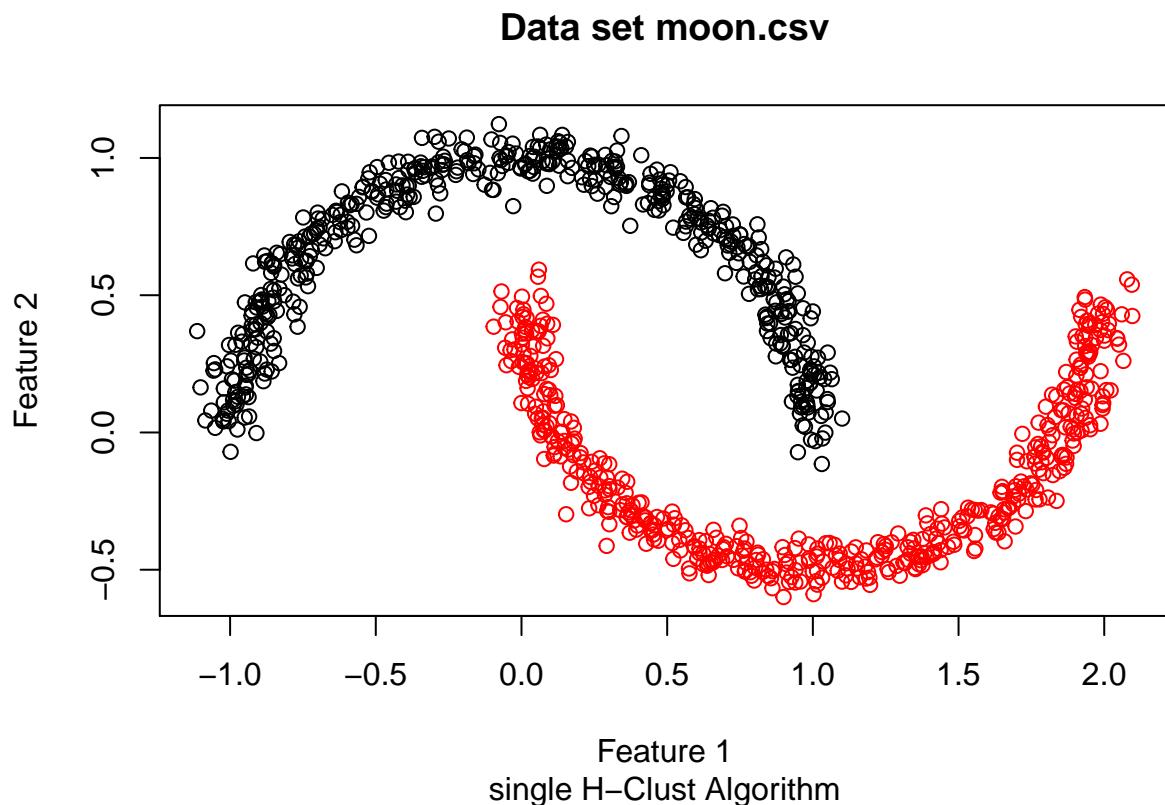
Necesitamos calcular la **matriz de distancias** antes de aplicar el algoritmo.

```
input_hierarchical_moon = df_moon  
input_hierarchical_moon$V3 = NULL  
input_hierarchical_moon = as.matrix(input_hierarchical_moon)  
hierarchical_distance_moon = dist(input_hierarchical_moon)
```

H-Clust Single

Generando el modelo

```
model_hierarchical_single_moon = eval_hclust(  
  distance = hierarchical_distance_moon, mode = "single",  
  centroids = 2, input = input_hierarchical_moon, dataname = "moon.csv")
```



Generando la matriz de confusión

```
table_model_hierarchical_single_moon = order_confusion_matrix(  
  confusion_matrix(df_moon$V3, model_hierarchical_single_moon))  
table_model_hierarchical_single_moon
```

```
##      PC1 PC2  
## TC1 500  0  
## TC2  0 500
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_single_moon, nrow(df_moon))
```

```
## Tasa de acierto: 100.000000  
## Tasa de fallo: 0.000000
```

Conclusión

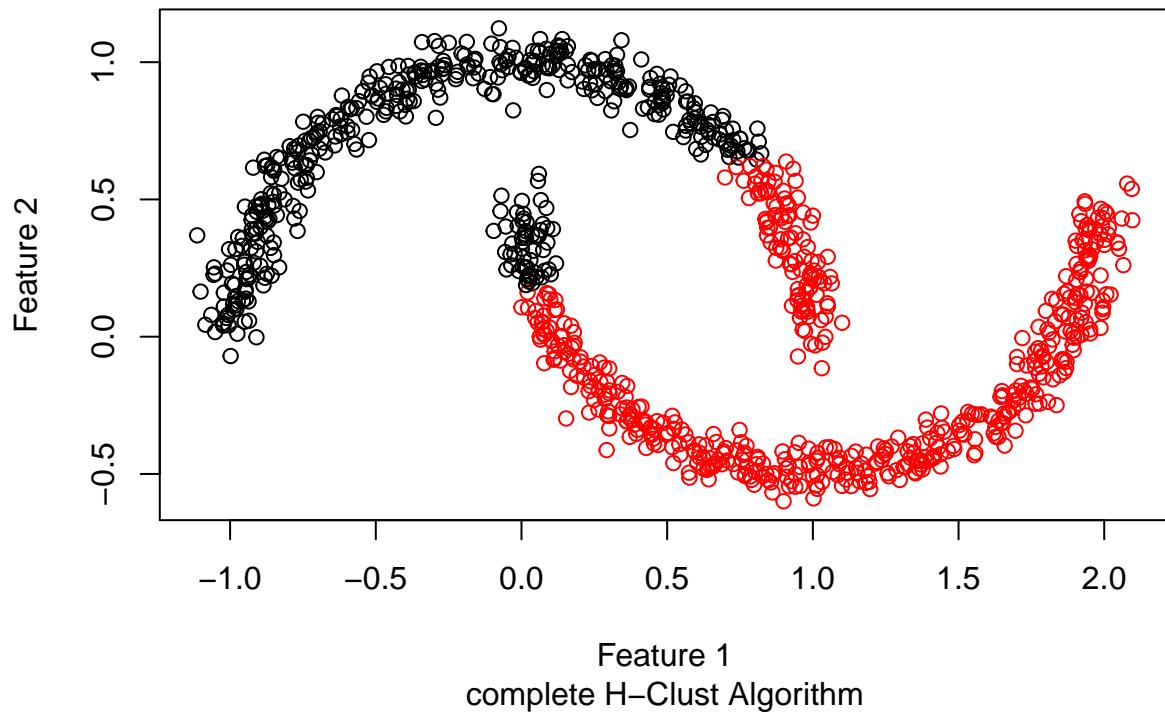
Como se puede apreciar este algoritmo tiene una precisión del 100%, debido a su naturaleza de ajustarse a formas no circulares o cilíndricas.

H-Clust Complete

Generando el modelo

```
model_hierarchical_complete_moon = eval_hclust(  
  distance = hierarchical_distance_moon, mode = "complete",  
  centroids = 2, input = input_hierarchical_moon, dataname = "moon.csv")
```

Data set moon.csv



Generando la matriz de confusión

```
table_model_hierarchical_complete_moon = order_confusion_matrix(  
  confusion_matrix(df_moon$V3, model_hierarchical_complete_moon))  
table_model_hierarchical_complete_moon
```

```
##      PC1 PC2  
##  TC1  393 107  
##  TC2   57 443
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_complete_moon, nrow(df_moon))
```

```
## Tasa de acierto: 83.600000  
## Tasa de fallo: 16.400000
```

Conclusión

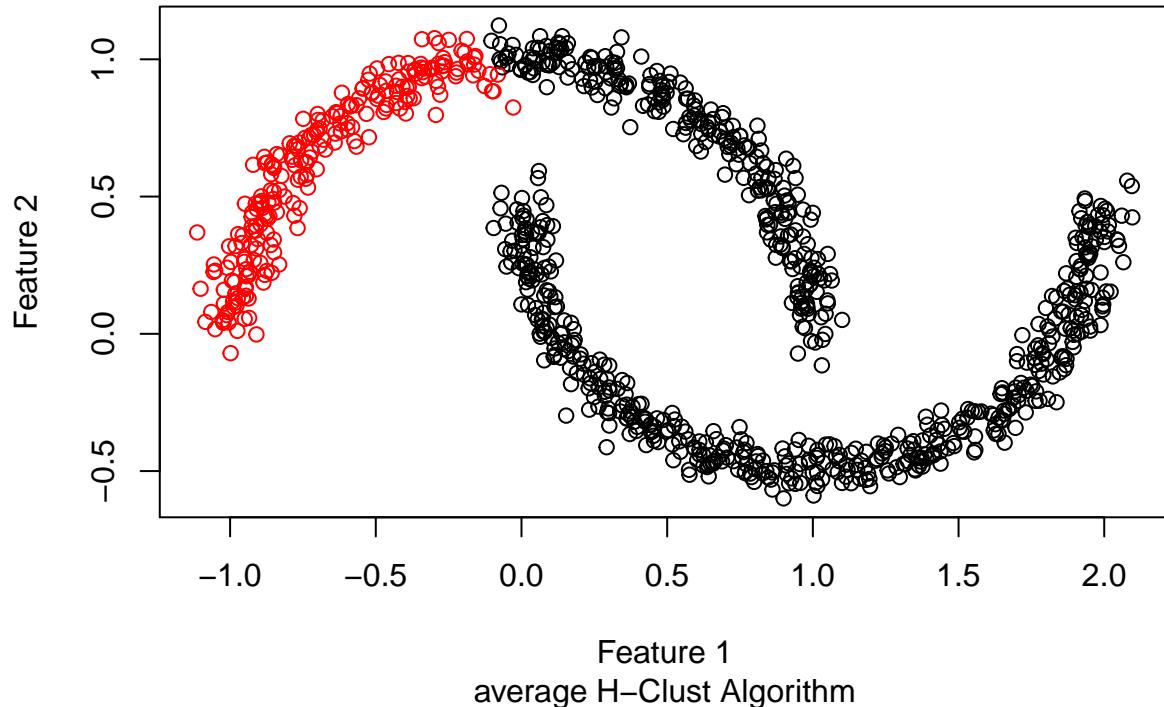
H-Clust Complete se comporta de manera similar a K-Medias, por lo que tiene un desempeño que deja mucho que desear.

H-Clust Average

Generando el modelo

```
model_hierarchical_average_moon = eval_hclust(  
  distance = hierarchical_distance_moon, mode = "average",  
  centroids = 2, input = input_hierarchical_moon, dataname = "moon.csv")
```

Data set moon.csv



Generando la matriz de confusión

```
table_model_hierarchical_average_moon = order_confusion_matrix(confusion_matrix(df_moon$V3, model_hiera  
table_model_hierarchical_average_moon
```

```
##      PC1 PC2  
##  TC1  234 266  
##  TC2     0 500
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_average_moon, nrow(df_moon))
```

```
## Tasa de acierto: 73.400000  
## Tasa de fallo: 26.600000
```

Conclusión

H-Clust average, tampoco muestra un buen rendimiento, debido a que las estructuras a la que mejor se ajusta este algoritmo es a aquellas que son circulares son patrones desordenados en los bordes.

Conclusión general

Por la estructura de semi-luna del conjunto de datos, el algoritmo **H-Clust Single** es el único que se ajusta bien al modelo, todos los demás algoritmos no generan un buen modelo.

good_luck.csv

Cargando el conjunto de datos

```
df_good_luck = read.csv("../data/good_luck.csv", header = F)
```

Pre-procesamiento

Hagamos un resumen rápido del conjunto de datos.

```
dim(df_good_luck)
```

```
## [1] 1000 11
```

Podemos ver que presenta mil (1000) filas y once (columnas), en la que la última corresponde a la columna de clases.

```
summary(df_good_luck$V11)
```

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

Podemos observar cómo el rango de los valores de dicha columna está en [0,1] y queremos colocarlas en el rango [1,2].

```
df_good_luck = pre_processing_especial(df_good_luck)
```

```
summary(df_good_luck$V11)
```

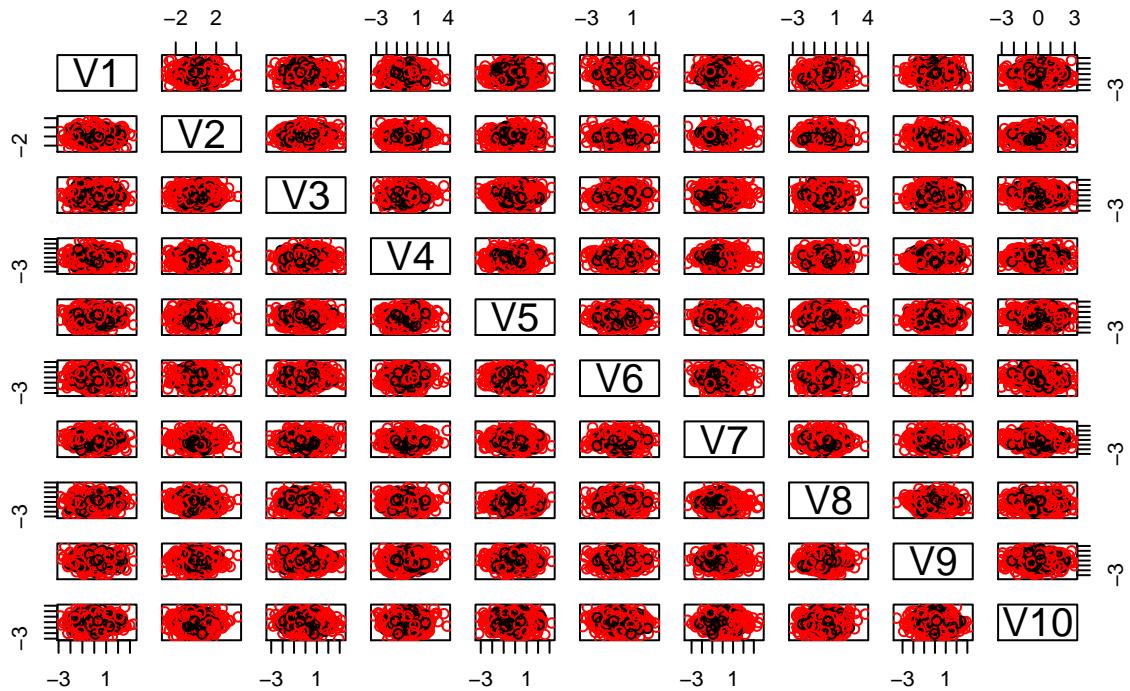
```
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.  
## 1.000 1.000 1.000 1.487 2.000 2.000
```

Ahora que tenemos el conjunto de datos podemos graficarlo.

Visualizando el conjunto de datos

```
plot(df_good_luck[,1:10], col = df_good_luck$V11,  
      main = "Matriz de Dispersion good_luck.csv")
```

Matriz de Dispersion good_luck.csv



Lo primero que podemos destacar es el hecho de que este conjunto de datos tiene diez (10) dimensiones, y que poder ver a simple vista cuál es la estructura del conjunto de datos es bastante complicado. Adicionalmente podemos ver cómo en cada una de los gráficos de las vistas, los puntos se solapan entre si, no dejando fronteras bien definidas entre cada clase. sin embargo parece ser que la estructura es de forma circular, por la forma en que aparece en la matriz de dispersión.

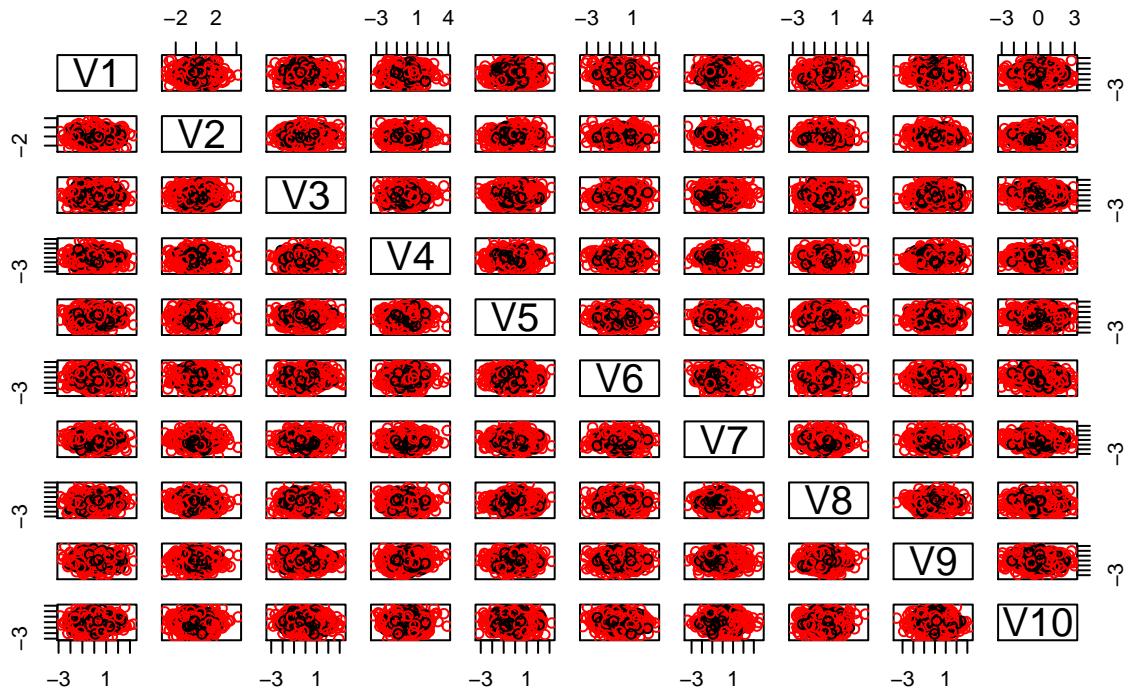
Aplicando los algoritmos

K-Medias

Generando el modelo

```
model_kmeans_good_luck = eval_kmeans_especial(  
    df = df_good_luck, cstart = 1, cfinish = 10,  
    k = 2, dataname = "good_luck.csv")
```

Matriz de Dispersion good_luck.csv



Generando la matriz de confusión

```
table_model_kmeans_good_luck = order_confusion_matrix(  
  confusion_matrix(df_good_luck$V11, model_kmeans_good_luck$cluster))  
table_model_kmeans_good_luck
```

```
##      PC1 PC2  
## TC1 274 239  
## TC2 232 255
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_kmeans_good_luck, nrow(df_good_luck))
```

```
## Tasa de acierto: 52.900000  
## Tasa de fallo: 47.100000
```

Conclusión

El algoritmo presenta sólo 50% de acierto, aunque a priori el conjunto de datos parece ser esférico el hecho de que posiblemente estén no separados de manera notable hace que se equivoque en la mitad de las veces.

H-Clust

Necesitamos calcular la **matriz de distancias** antes de aplicar el algoritmo.

```

input_hierarchical_good_luck = df_good_luck
input_hierarchical_good_luck$V11 = NULL
input_hierarchical_good_luck = as.matrix(input_hierarchical_good_luck)
hierarchical_distance_good_luck = dist(input_hierarchical_good_luck)

```

H-Clust Single

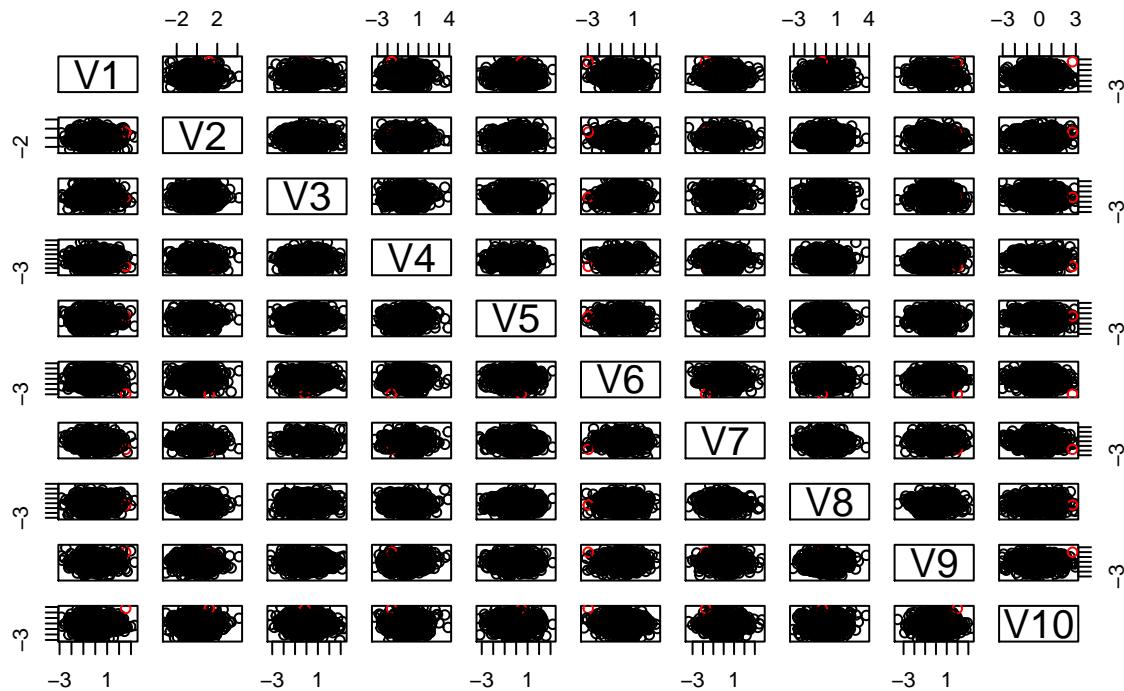
Generando el modelo

```

model_hierarchical_single_good_luck = eval_hclust_especial(
  distance = hierarchical_distance_good_luck, mode = "single",
  centroids = 2, input = input_hierarchical_good_luck, dataname = "good_luck.csv",
  cfinish = 10)

```

Data set good_luck.csv



Generando la matriz de confusión

```

table_model_hierarchical_single_good_luck = order_confusion_matrix(
  confusion_matrix(df_good_luck$V11, model_hierarchical_single_good_luck))
table_model_hierarchical_single_good_luck

```

```

##      PC1 PC2
## TC1 513   0
## TC2 486   1

```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_single_good_luck,  
                           nrow(df_good_luck))  
  
## Tasa de acierto: 51.400000  
## Tasa de fallo: 48.600000
```

Conclusión

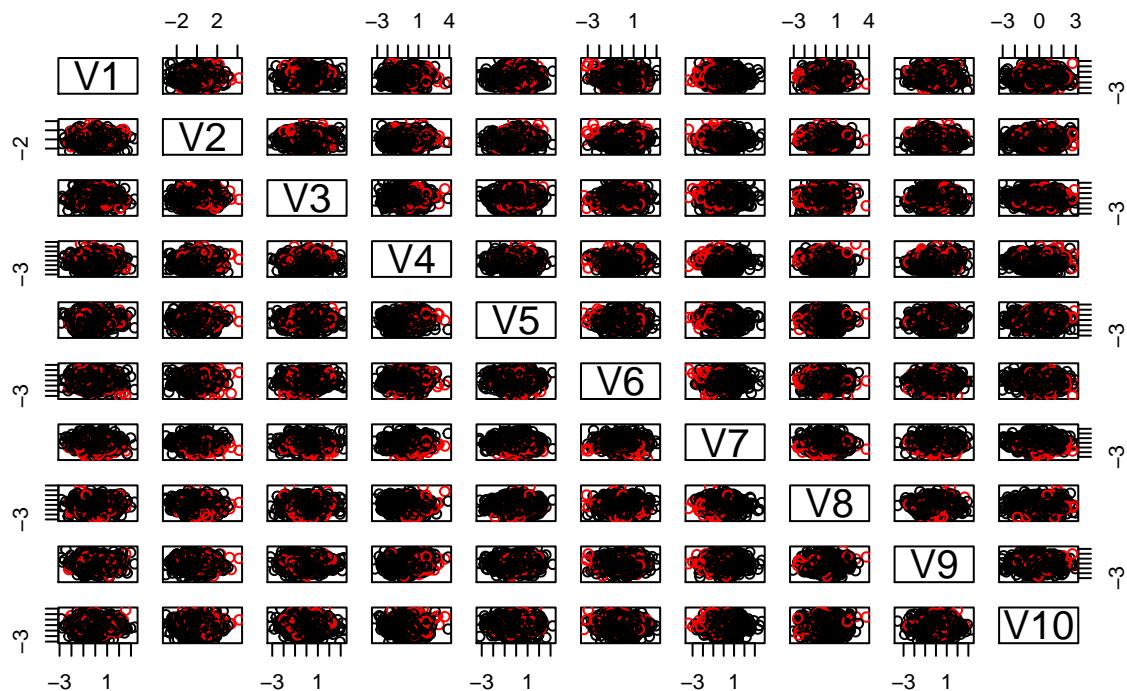
La tasa de acierto es de 51% sin embargo se puede ver cómo se clasificaron todos los elementos menos uno como un sólo cluster, lo que nos da a entender que por la naturaleza del algoritmo de buscar aquellos clusters que poseen formas no circulares y con límites bien definidos, el conjunto de datos presenta justamente lo contrario, apoyando nuestro análisis exploratorio.

H-Clust Complete

Generando el modelo

```
model_hierarchical_complete_good_luck = eval_hclust_especial(  
  distance = hierarchical_distance_good_luck, mode = "complete",  
  centroids = 2, input = input_hierarchical_good_luck, dataname = "good_luck.csv",  
  cfinish = 10)
```

Data set good_luck.csv



Generando la matriz de confusión

```
table_model_hierarchical_complete_good_luck = order_confusion_matrix(  
  confusion_matrix(df_good_luck$V11, model_hierarchical_complete_good_luck))  
table_model_hierarchical_complete_good_luck
```

```
##      PC1 PC2  
## TC1 467  46  
## TC2 409  78
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_complete_good_luck,  
  nrow(df_good_luck))
```

```
## Tasa de acierto: 54.500000  
## Tasa de fallo: 45.500000
```

Conclusión

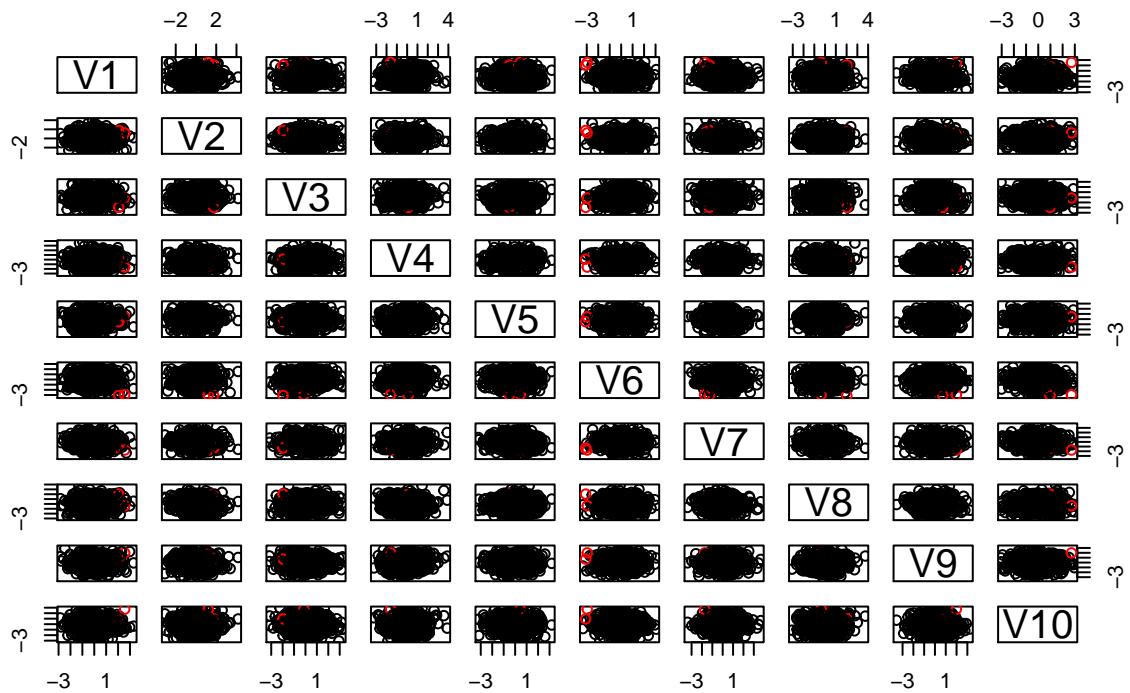
El algoritmo se comporta mejor que los otros dos anteriores, sin embargo sigue teniendo el mismo comportamiento de querer colocar la mayoría de los puntos en un sólo cluster, debido a la ausencia de una frontera bien definida entre las clases.

H-Clust Average

Generando el modelo

```
model_hierarchical_average_good_luck = eval_hclust_especial(  
  distance = hierarchical_distance_good_luck, mode = "average",  
  centroids = 2, input = input_hierarchical_good_luck, dataname = "good_luck.csv",  
  cfinish = 10)
```

Data set good_luck.csv



Generando la matriz de confusión

```
table_model_hierarchical_average_good_luck = order_confusion_matrix(
  confusion_matrix(df_good_luck$V11, model_hierarchical_average_good_luck))
table_model_hierarchical_average_good_luck
```

```
##      PC1 PC2
## TC1  513   0
## TC2  485   2
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(
  table_model_hierarchical_average_good_luck, nrow(df_good_luck))
```

```
## Tasa de acierto: 51.500000
## Tasa de fallo: 48.500000
```

Conclusión

Se comporta parecido a **H-Clust Single**, con un error del 50% pero tratando de colocar todos los elementos dentro de un mismo cluster.

Conclusión general

Este conjunto de datos posee una estructura muy complicada y difícil de poder separar, ninguno de los algoritmos que se probaron funciona de buena manera, esto debido a que las clases no poseen separaciones

marcadas entre si y que posiblemente las clases estén superpuestas una sobre otra, ya que el algoritmo de **K-Medias** se equivocó en la mitad de los casos pero tuvo una distribución más equitativa de la matriz de confusión con respecto a la cardinalidad de las columnas de la matriz.

Análisis con clases reales

Consta de tres conjuntos de datos:

- h.csv
- s.csv
- help.csv

Vamos a empezar a trabajar en ellos.

h.csv

Cargando el conjunto de datos

```
df_h = read.csv("../data/h.csv", header = F)
```

Pre-procesamiento

Veamos el rango de los valores de las clases de **h.csv**.

```
summary(df_h$V4)
```

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

Queremos colocar los valores de la columna de clases en el rango [1,11] en vez del [4, 15].

```
df_h$V5 = floor(df_h$V4) - min(floor(df_h$V4)) + 1
```

Visualizando el conjunto de datos

Matriz de dispersión

```
plot(df_h[1:3], col= get_colors(11, df_h),
      main = "Matriz Dispersion h.csv")
```

Matriz Dispersion h.csv

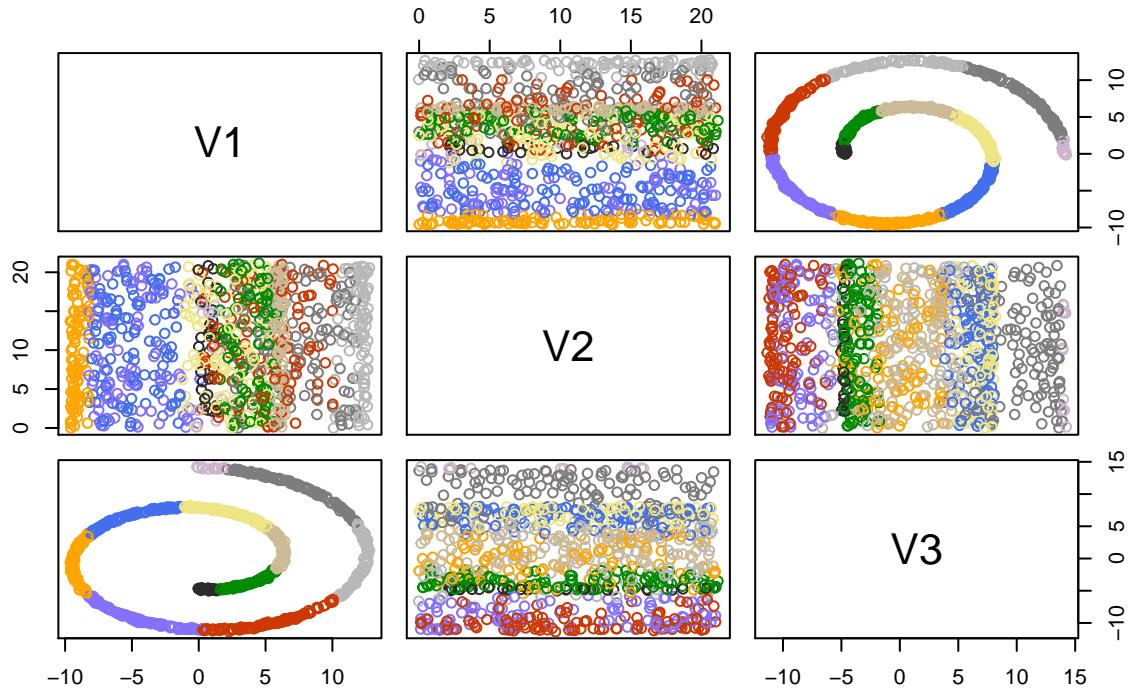
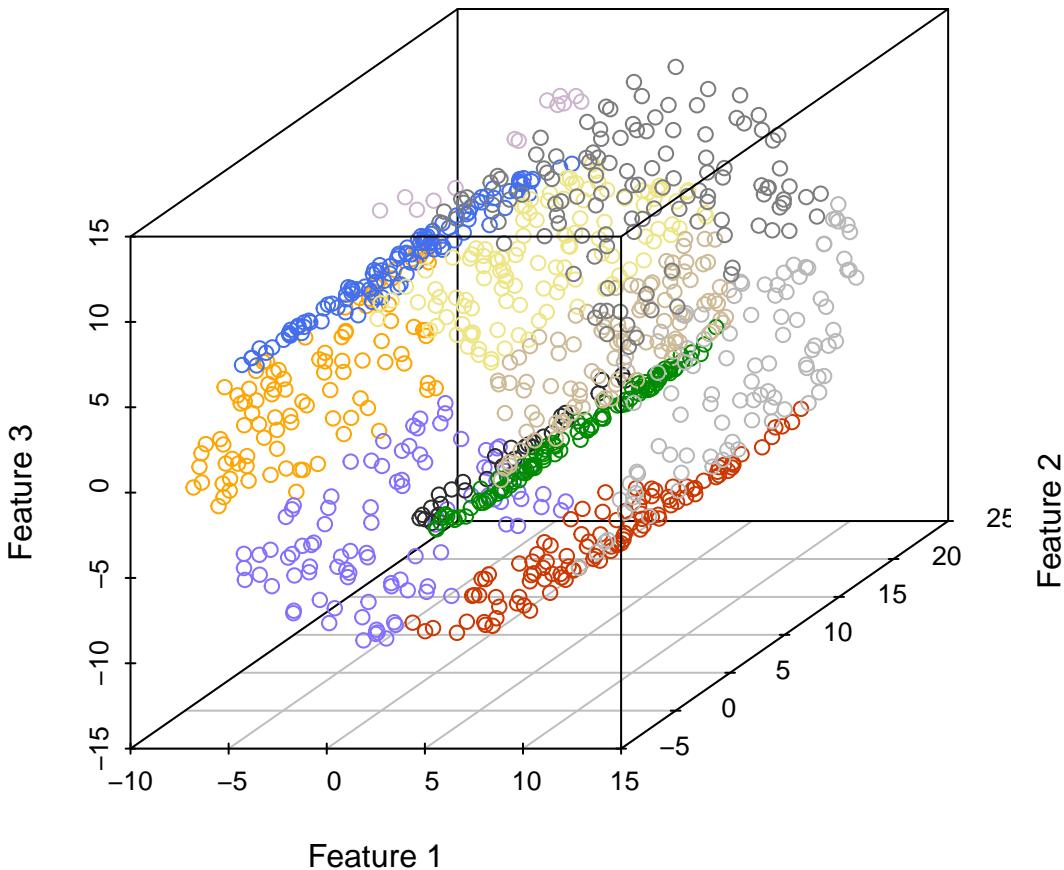


Gráfico 3D

```
scatterplot3d(df_h$V1, df_h$V2, df_h$V3,
              color = get_colors(11, df_h),
              xlab = "Feature 1", ylab = "Feature 2", zlab = "Feature 3",
              main= "Data Set h.csv")
```

Data Set h.csv



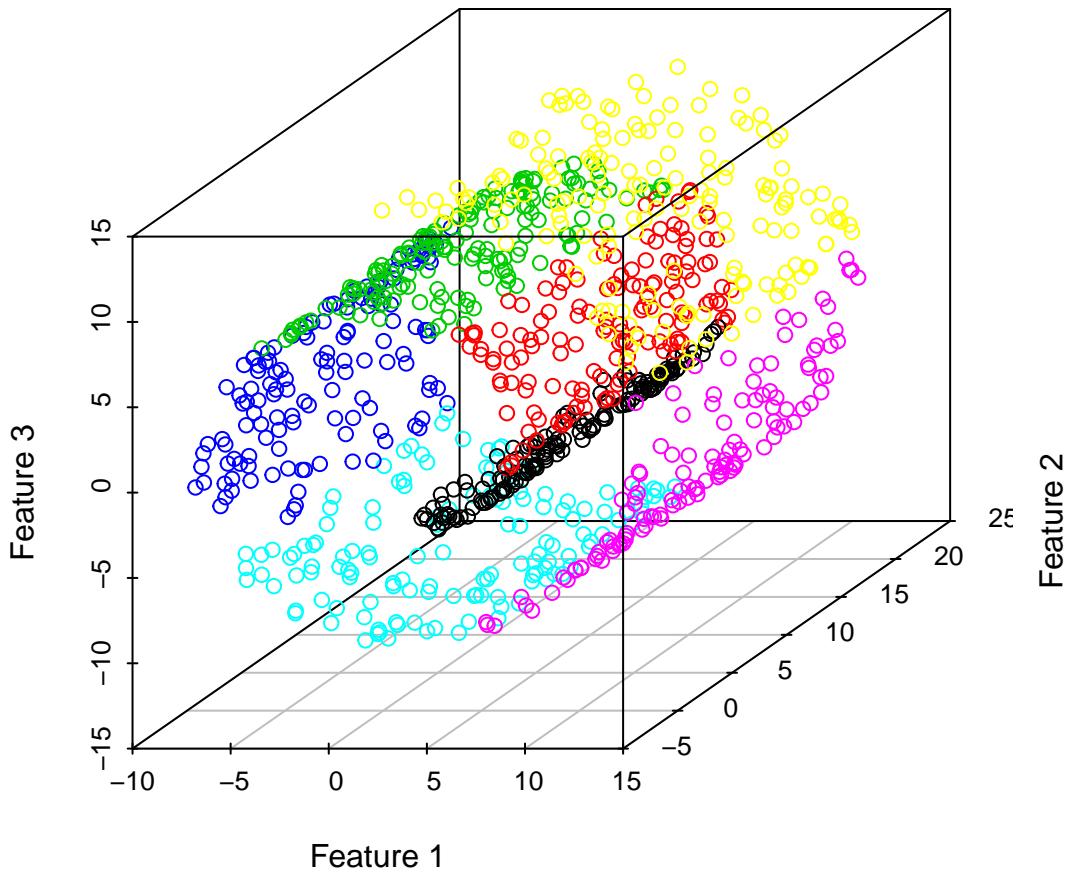
Se pueden observar once (11) colores, donde cada color corresponde a una clase independiente, sin embargo, existen clases que contienen pocos individuos, lo que se hará, es llevar las clases a valores en el rango [1,7] para representar de mejor manera el conjunto de datos.

```
df_h$V5 = NULL  
df_h$class = 7*(df_h$V4-min(df_h$V4))/(max(df_h$V4)-min(df_h$V4)) + 1  
df_h$class = floor(df_h$class)  
df_h$class[df_h$class == 8] = 7
```

Luego de que se hace la re-escala de las clases, el conjunto de datos queda de la siguiente manera:

```
scatterplot3d(df_h$V1, df_h$V2, df_h$V3, color = df_h$class,  
             xlab = "Feature 1", ylab = "Feature 2", zlab = "Feature 3",  
             main= "Data Set h.csv")
```

Data Set h.csv



Ahora, que el conjunto está mejor representando, podemos empezar a aplicar los algoritmos.

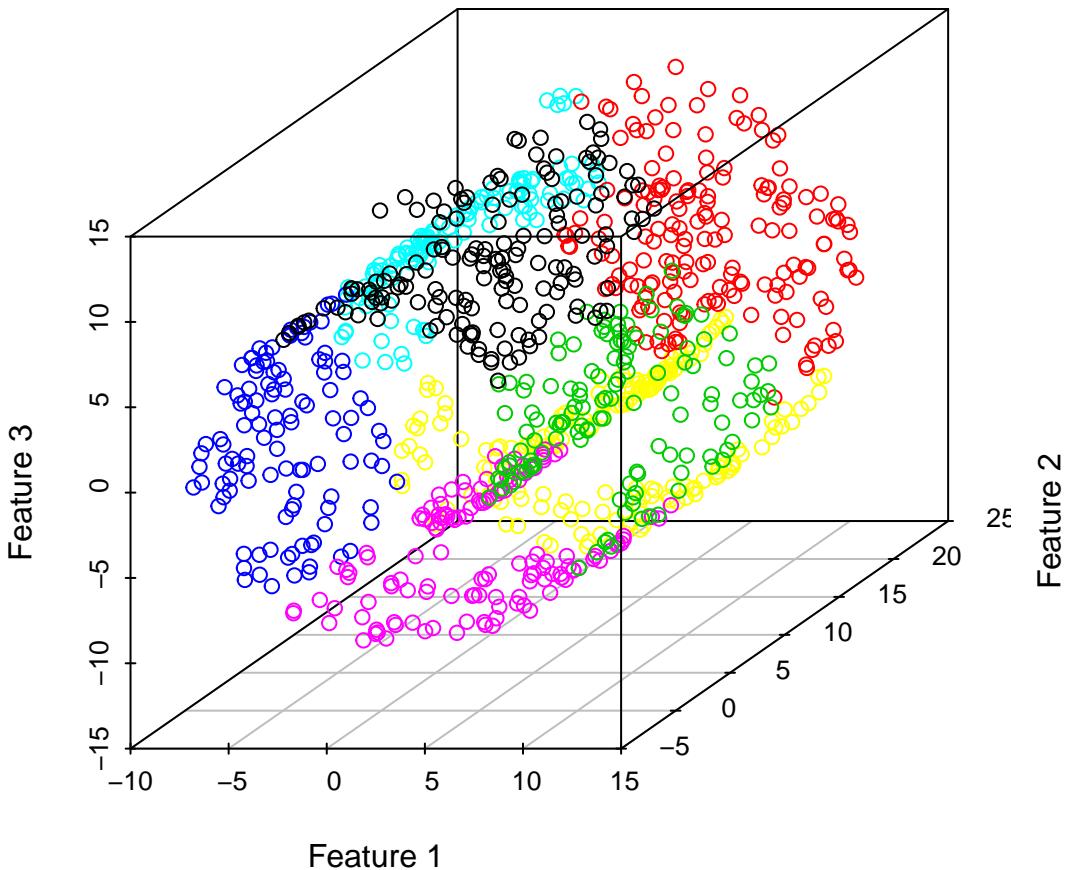
Aplicando los algoritmos

K-Medias

Generando el modelo

```
model_kmeans_h = eval_kmeans_3D(df = df_h, cstart = 1, cfinish = 3,  
                                k = 7, dataname = "h.csv")
```

Data Set h.csv



Generando la matriz de confusión

```
table_model_kmeans_h = order_confusion_matrix(  
  confusion_matrix(df_h$class, model_kmeans_h$cluster))  
table_model_kmeans_h
```

```
##      PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## TC1    74  21   0   0  51   0   0  
## TC2     4  46   0   0   0  30  63  
## TC3     0   0  74   5   0  67  14  
## TC4     7   0  43  84   0   0   0  
## TC5    56   0   0  24  57   0   0  
## TC6    34  43   0   0  24   0  17  
## TC7     0  19   5   0   0  61  77
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_kmeans_h, nrow(df_h))
```

```
## Tasa de acierto: 41.200000  
## Tasa de fallo: 58.800000
```

Conclusión

Por la forma del conjunto de datos, que no es cilíndrico, entonces el algoritmo de K-Medias no se comporta de buena manera teniendo poca tasa de acierto.

H-Clust

Necesitamos calcular la **matriz de distancias** antes de aplicar el algoritmo.

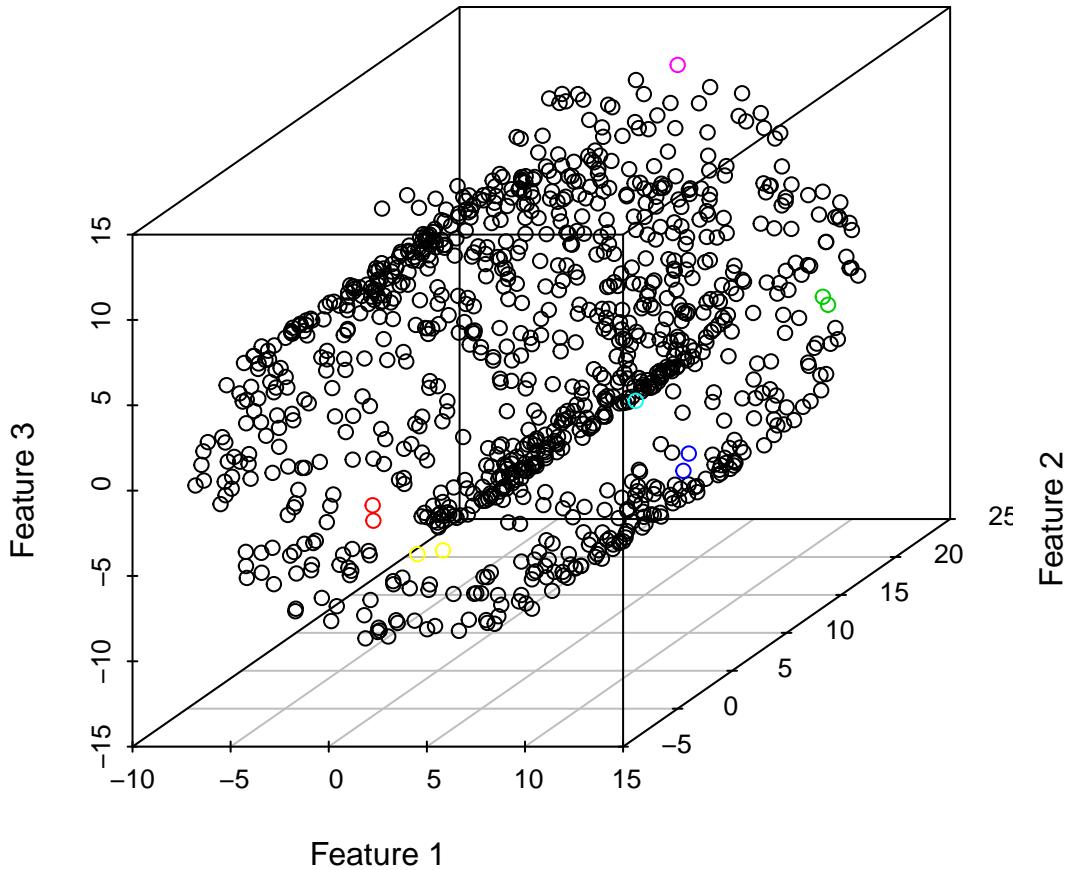
```
input_hierarchical_h = df_h
input_hierarchical_h$V4 = NULL
input_hierarchical_h$class = NULL
input_hierarchical_h = as.matrix(input_hierarchical_h)
hierarchical_distance_h = dist(input_hierarchical_h)
```

H-Clust Single

Generando el modelo

```
model_hierarchical_single_h = eval_hclust_3D(
  distance = hierarchical_distance_h, mode = "single", centroids = 7,
  input = input_hierarchical_h, dataname = "h.csv")
```

Data set h.csv



single H-Clust Algorithm

Generando la matriz de confusión

```
table_model_hierarchical_single_h = order_confusion_matrix(  
  confusion_matrix(df_h$class, model_hierarchical_single_h))  
table_model_hierarchical_single_h
```

```
##      PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## TC1    0   0   0   0   0   0 146  
## TC2    0   0   0   0   0   0 143  
## TC3    0   0   0   0   0   0 160  
## TC4    0   0   0   0   0   0 134  
## TC5    0   2   0   0   2   0 133  
## TC6    2   0   1   0   0   2 113  
## TC7    0   0   0   1   0   0 161
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_single_h, nrow(df_h))
```

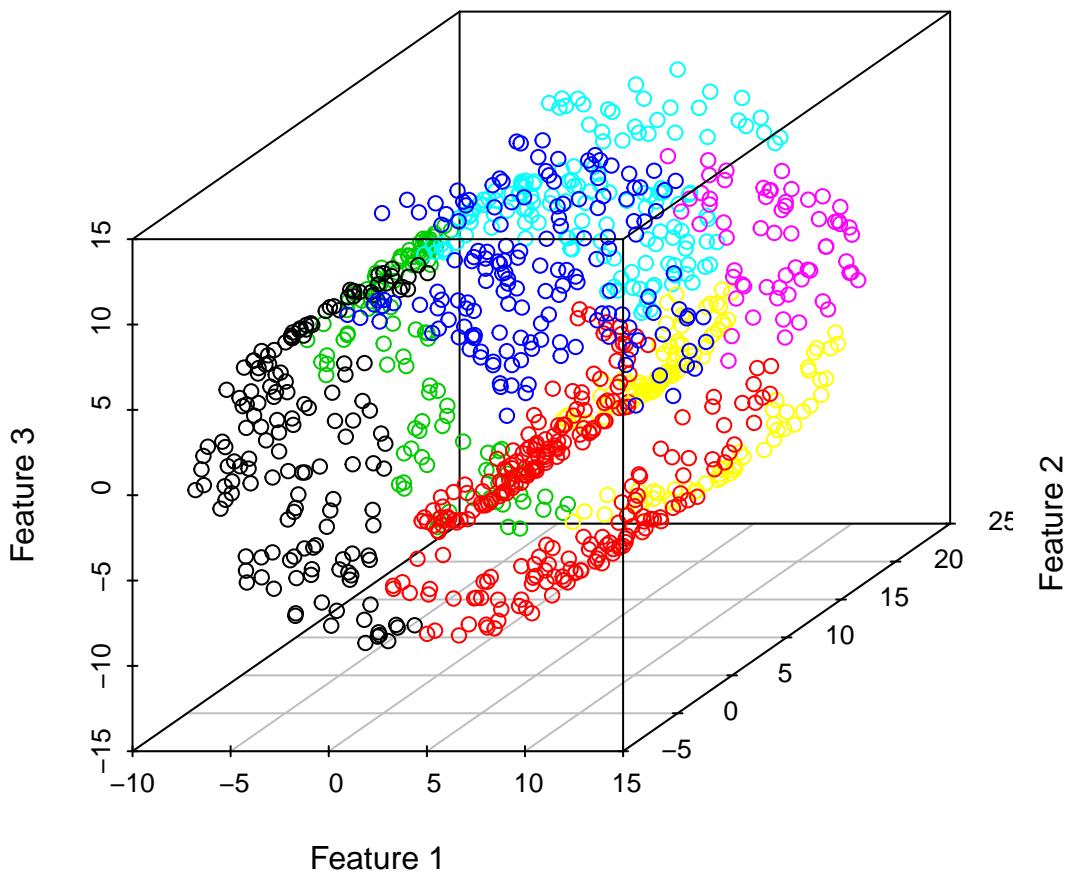
```
## Tasa de acierto: 16.500000
## Tasa de fallo: 83.500000
```

H-Clust Complete

Generando el modelo

```
model_hierarchical_complete_h = eval_hclust_3D(
  distance = hierarchical_distance_h, mode = "complete", centroids = 7,
  input = input_hierarchical_h, dataname = "h.csv")
```

Data set h.csv



Generando la matriz de confusión

```
table_model_hierarchical_complete_h = order_confusion_matrix(  
  confusion_matrix(df_h$class, model_hierarchical_complete_h))  
table_model_hierarchical_complete_h
```

```
##      PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## TC1  92   0   0   0  54   0   0  
## TC2  42   0  43   0  22   0  36  
## TC3   0  17  71  32   0   0  40  
## TC4   0  55   0  79   0   0   0  
## TC5  45  35   0  44  13   0   0  
## TC6  68   0   0   0  37  10   3  
## TC7   0   0  31   0   0  52  79
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_complete_h, nrow(df_h))
```

```
## Tasa de acierto: 34.400000  
## Tasa de fallo: 65.600000
```

Conclusión

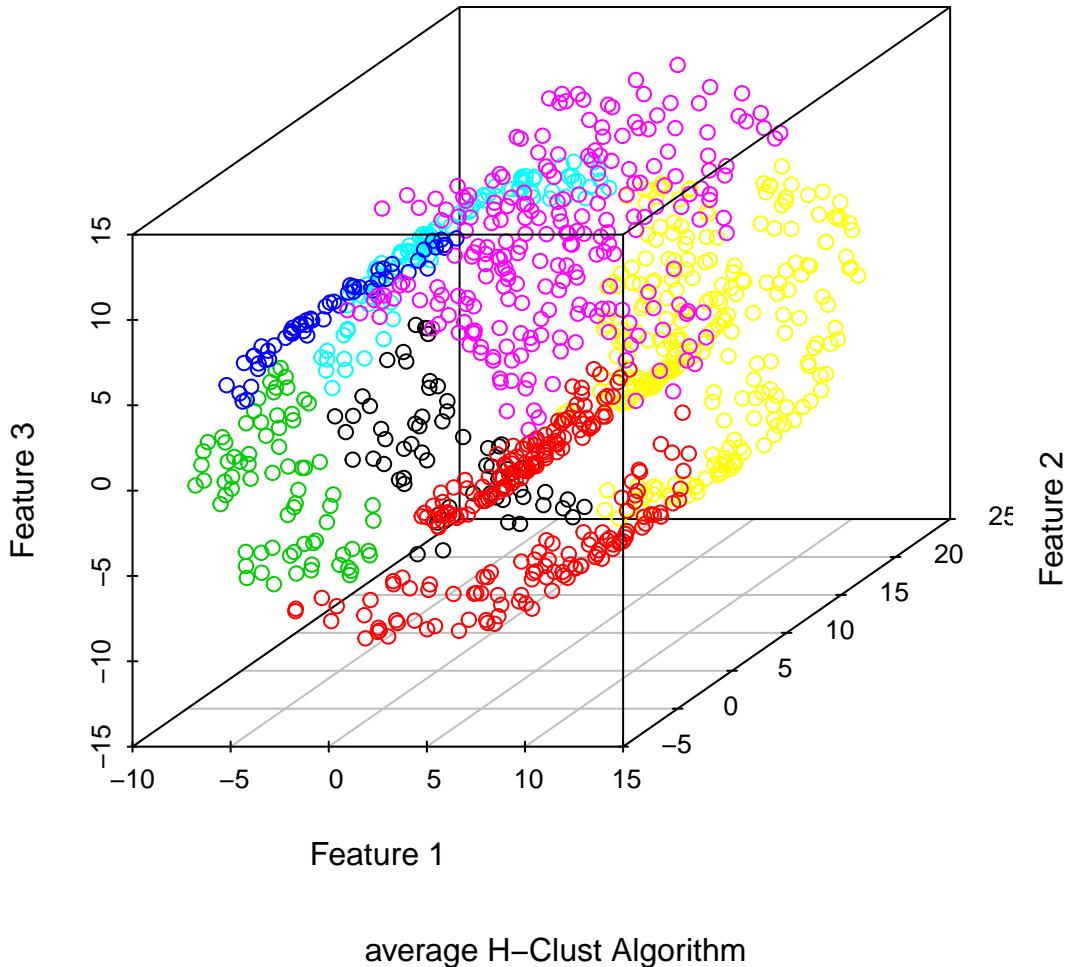
H-Clust Complete se comporta de manera similar a K-Medias, tratando de buscar figuras y/o superficies cilíndricas, caso que no se presenta en este conjunto de datos, por lo mismo presenta más fallos que aciertos.

H-Clust Complete

Generando el modelo

```
model_hierarchical_average_h = eval_hclust_3D(  
  distance = hierarchical_distance_h, mode = "average", centroids = 7,  
  input = input_hierarchical_h, dataname = "h.csv")
```

Data set h.csv



Generando la matriz de confusión

```
table_model_hierarchical_average_h = order_confusion_matrix(  
  confusion_matrix(df_h$class, model_hierarchical_average_h))  
table_model_hierarchical_average_h
```

```
##      PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## TC1    91  55   0   0   0   0   0  
## TC2    23  74   0   0   0   0  46  
## TC3     0   6  56   0   0  37  61  
## TC4     0   0  43  46  23  22   0  
## TC5    55  14   0  28  40   0   0  
## TC6    50  65   0   0   0   0   3  
## TC7     0  39   0   0   0   0 123
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_average_h, nrow(df_h))

## Tasa de acierto: 43.000000
## Tasa de fallo: 57.000000
```

Conclusión

Este modelo es el que mejor se ajusta de todos los evaluados, sin embargo sigue presentando un pobre desempeño, debido a que la forma del conjunto de datos no corresponde a superficies cilíndricas.

Conclusión general

Ninguno de los algoritmos aplicados funciona de una manera correcta para este conjunto de datos debido a la naturaleza del mismo, presenta una forma en espiral que los algoritmos de **K-Medias** y **H-Clust** no saben interpretar, presentando un pobre rendimiento. Podrían evaluarse otros algoritmos que se adaptaran mejor a este tipo de problemas tales como **Spectral CLustering**.

S.CSV

Cargando el conjunto de datos

```
df_s = read.csv("../data/s.csv", header = F)
```

Pre-procesamiento

Veamos el rango de los valores de las clases de **h.csv**.

```
summary(df_s$V4)
```

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

Podemos ver que los rangos de valores de las clases están en el rango [-5, 5]. Ahora veamos el comportamiento del conjunto de datos, pero para ello, debemos llevar los valores de la columna clase a un rango positivo entre [1,10]

```
df_s$V5 = floor(df_s$V4) - min(floor(df_s$V4)) +1
```

Ahora grafiquemos.

Matriz de Dispersion

```
plot(df_s[,1:3], col= get_colors(11, df_s),
      main = "Matriz Dispersion s.csv")
```

Matriz Dispersion s.csv

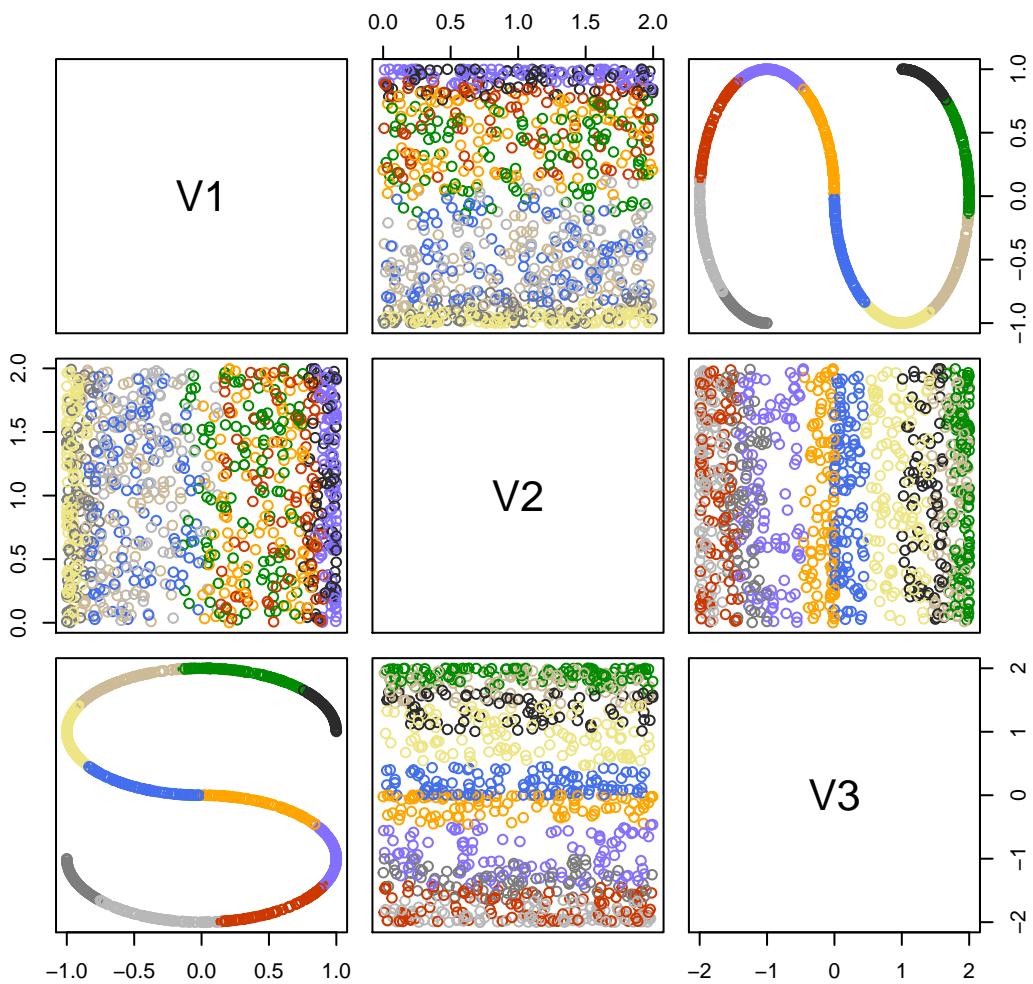
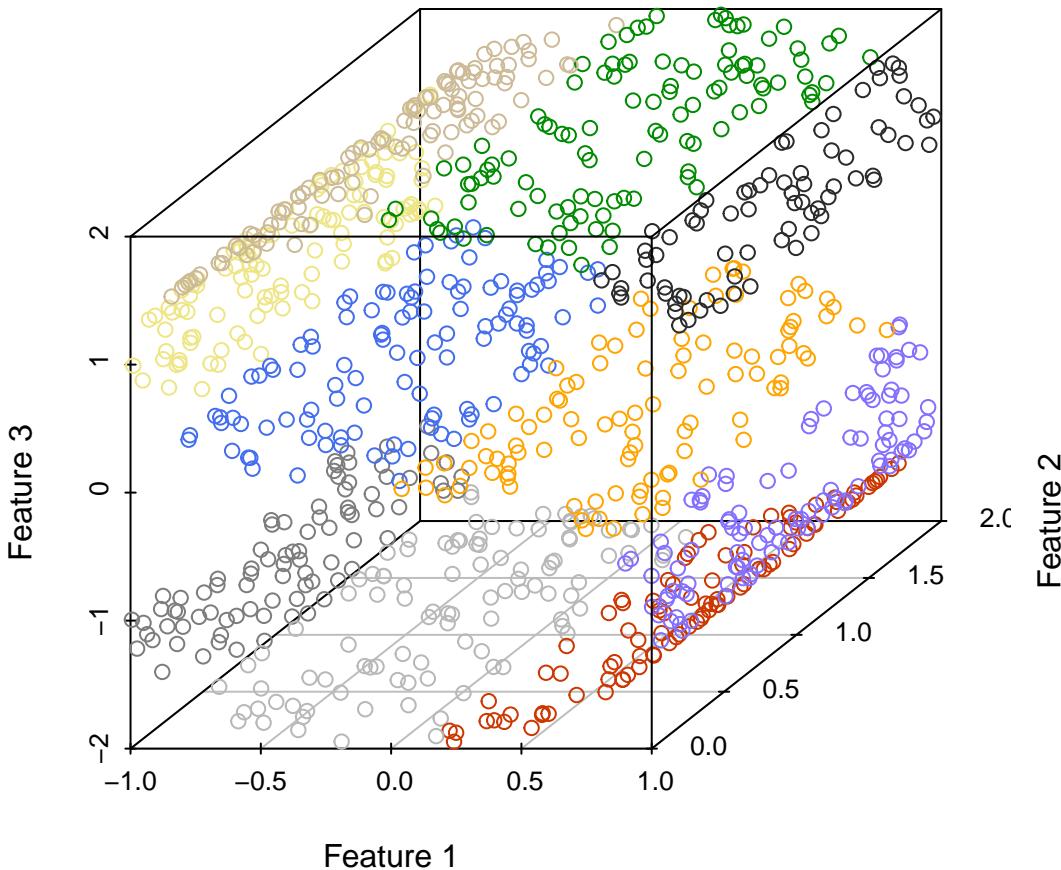


Gráfico 3D

```
scatterplot3d(df_s$V1, df_s$V2, df_s$V3, color = get_colors(11, df_s),
              type="p", xlab = "Feature 1", ylab = "Feature 2", zlab = "Feature 3",
              main= "Data Set s.csv")
```

Data Set s.csv



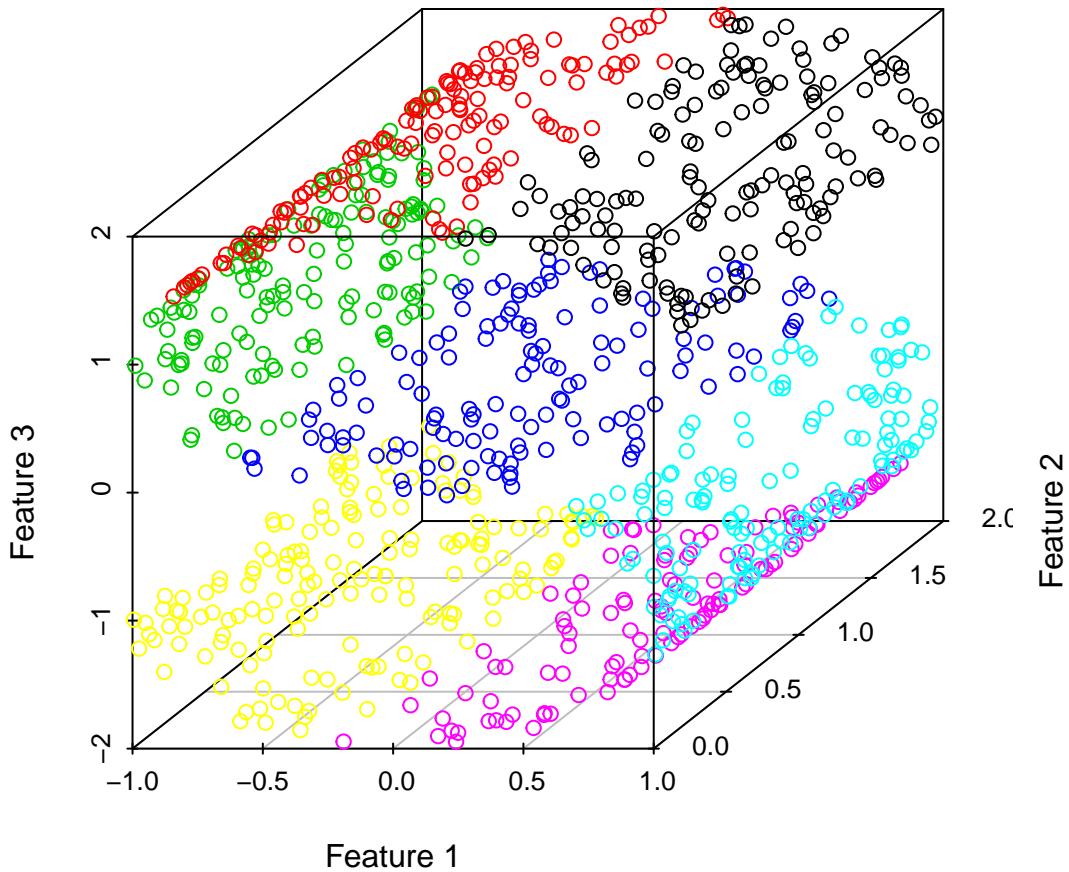
Podemos ver once (11) clases, que se presentan de manera muy similar al conjunto de datos **h.csv**, de igual manera, hay clases que constan de muy pocos individuos que vamos a compactar para llevarlas al rango [1, 7] para mejor representación del conjunto de datos.

```
df_s$V5 = NULL
df_s$class = 7*(df_s$V4-min(df_s$V4))/(max(df_s$V4)-min(df_s$V4)) + 1
df_s$class = floor(df_s$class)
df_s$class[df_s$class == 8] = 7
```

Una vez que hacemos el re-escalamiento de la columna clase, ahora el conjunto de datos luce de la siguiente manera:

```
scatterplot3d(df_s$V1, df_s$V2, df_s$V3, color = df_s$class,
  type="p", xlab = "Feature 1", ylab = "Feature 2", zlab = "Feature 3",
  main= "Data Set s.csv")
```

Data Set s.csv



Ahora, que el conjunto está mejor representando, podemos empezar a aplicar los algoritmos.

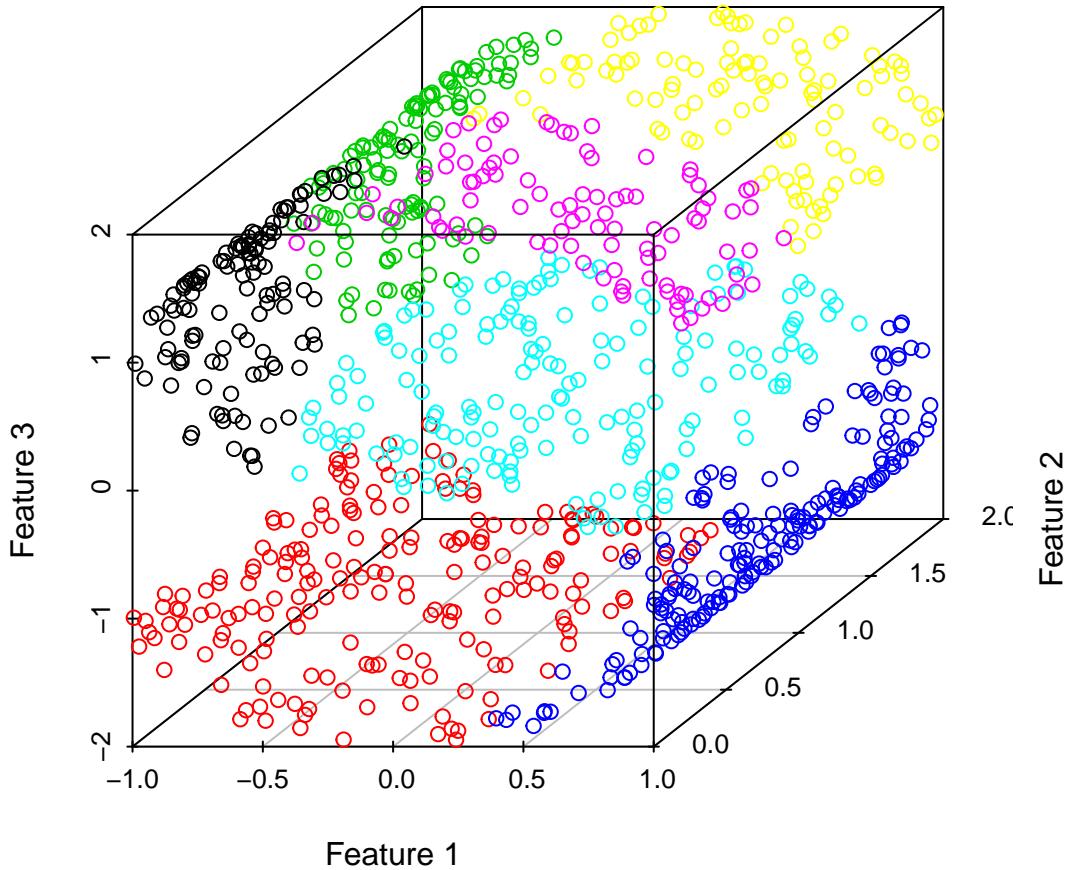
Aplicando los algoritmos

K-Medias

Generando el modelo

```
model_kmeans_s = eval_kmeans_3D(df = df_s, cstart = 1, cfinish = 3,
                                 k = 7, dataname = "s.csv")
```

Data Set s.csv



Generando la matriz de confusión

```
table_model_kmeans_s = order_confusion_matrix(confusion_matrix(df_s$class, model_kmeans_s$cluster))
table_model_kmeans_s
```

```
##      PC1 PC2 PC3 PC4 PC5 PC6 PC7
## TC1    81   0   0   0   0  65   0
## TC2    24  50  37   0   0  32   0
## TC3     0  74  79   7   0   0   0
## TC4     0   0    3 131   0   0   0
## TC5     0   0    0  31 106   0   0
## TC6     0   0    0   0  78   0  40
## TC7     0   0    0   0   0   0 162
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_kmeans_h, nrow(df_h))
```

```
## Tasa de acierto: 41.200000
## Tasa de fallo: 58.800000
```

Conclusión

Por la forma de “S” que presenta el conjunto de datos, muy alejado de las formas cilíndricas para las cuales K-Medias funciona de buena forma, el algoritmo presenta un pobre rendimiento en esta ocasión.

H-Clust

Necesitamos calcular la **matriz de distancias** antes de aplicar el algoritmo.

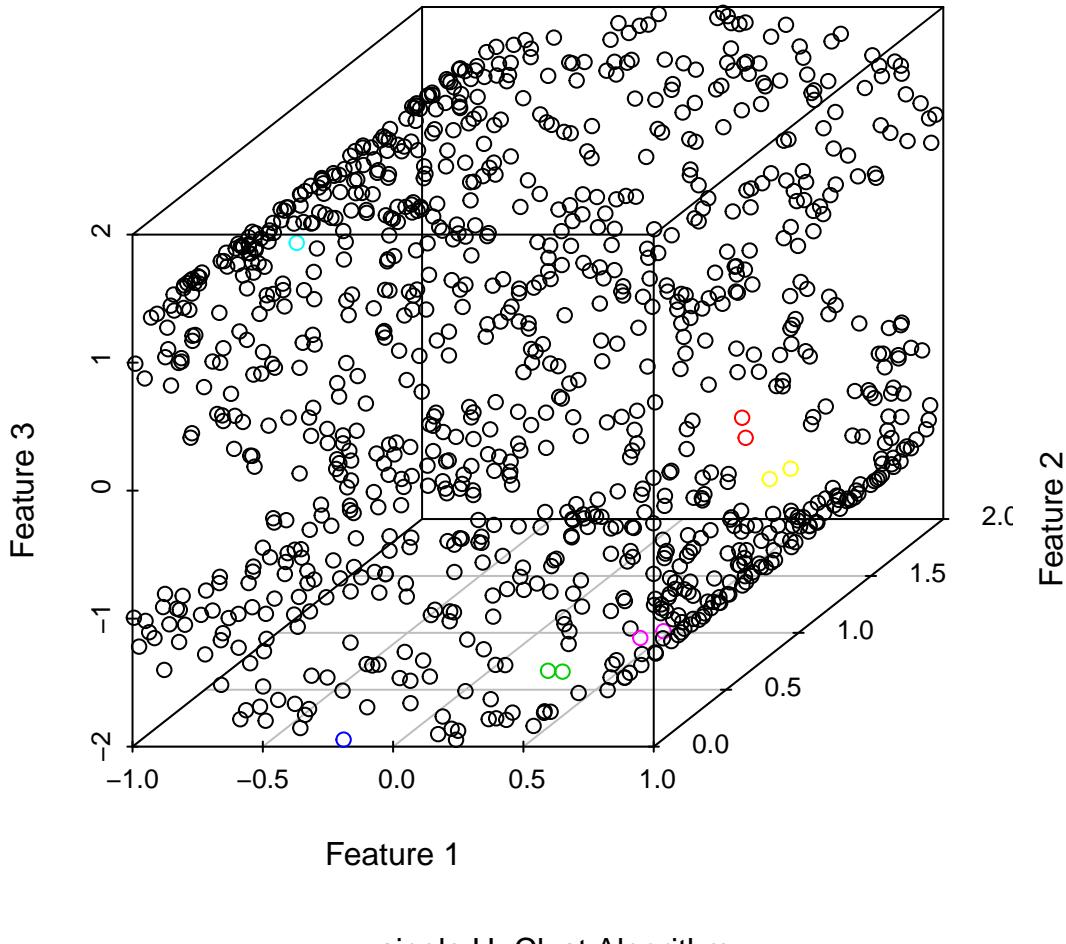
```
input_hierarchical_s = df_s
input_hierarchical_s$V4 = NULL
input_hierarchical_s$class = NULL
input_hierarchical_s = as.matrix(input_hierarchical_s)
hierarchical_distance_s = dist(input_hierarchical_s)
```

H-Clust Single

Generando el modelo

```
model_hierarchical_single_s = eval_hclust_3D(
  distance = hierarchical_distance_s, mode = "single", centroids = 7,
  input = input_hierarchical_s, dataname = "s.csv")
```

Data set s.csv



Feature 1

Feature 2

Feature 3

single H-Clust Algorithm

Generando la matriz de confusión

```
table_model_hierarchical_single_s = order_confusion_matrix(  
  confusion_matrix(df_s$class, model_hierarchical_single_s))  
table_model_hierarchical_single_s
```

```
##      PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## TC1    0   0   0   0   0   0 146  
## TC2    0   1   0   0   0   0 142  
## TC3    0   0   0   0   0   0 160  
## TC4    0   0   0   0   0   0 134  
## TC5    0   0   0   2   2   0 133  
## TC6    2   0   1   0   0   2 113  
## TC7    0   0   0   0   0   0 162
```

Evaluando la matriz de confusión

```

confusion_matrix_evaluation(table_model_hierarchical_single_s, nrow(df_s))

## Tasa de acierto: 16.70000
## Tasa de fallo: 83.30000

```

Conclusión

Pasa lo mismo que en **h.csv**, cómo los clusters están unidos, el algoritmo los toma todos como si fueran uno solo y tiene un pésimo rendimiento.

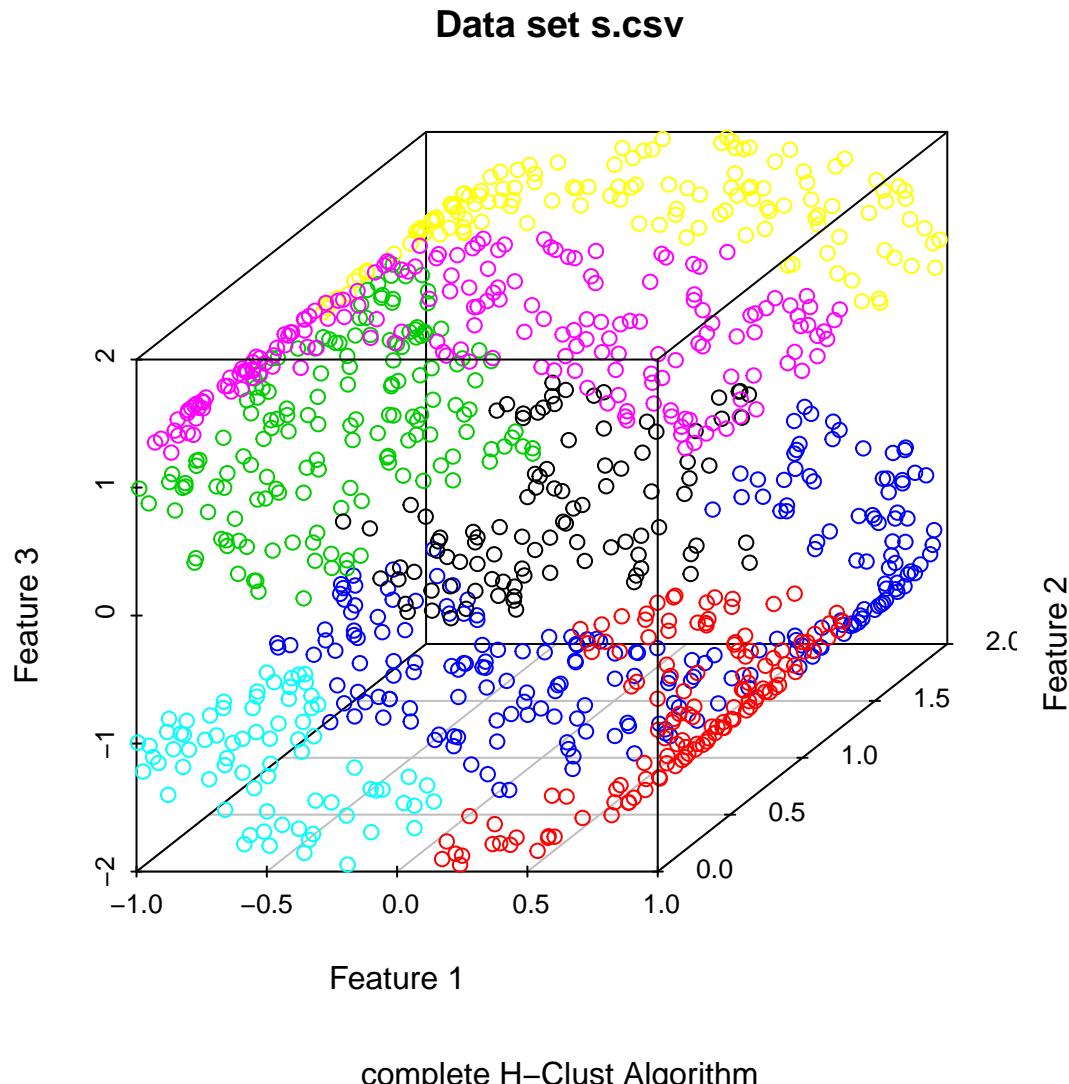
H-Clust Complete

Generando el modelo

```

model_hierarchical_complete_s = eval_hclust_3D(
  distance = hierarchical_distance_s, mode = "complete", centroids = 7,
  input = input_hierarchical_s, dataname = "s.csv")

```



Generando la matriz de confusión

```
table_model_hierarchical_complete_s = order_confusion_matrix(confusion_matrix(
  df_s$class, model_hierarchical_complete_s))
table_model_hierarchical_complete_s
```

```
##      PC1 PC2 PC3 PC4 PC5 PC6 PC7
## TC1   86  60   0   0   0   0   0
## TC2   82  61   0   0   0   0   0
## TC3   21  13 126   0   0   0   0
## TC4    0   0  30  93   0   0  11
## TC5    0   0   0   5  80   0  52
## TC6    0   0   0   0  50   3  65
## TC7    0   0   0   0   0  72  90
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_complete_s, nrow(df_s))
```

```
## Tasa de acierto: 53.900000
## Tasa de fallo: 46.100000
```

Conclusión

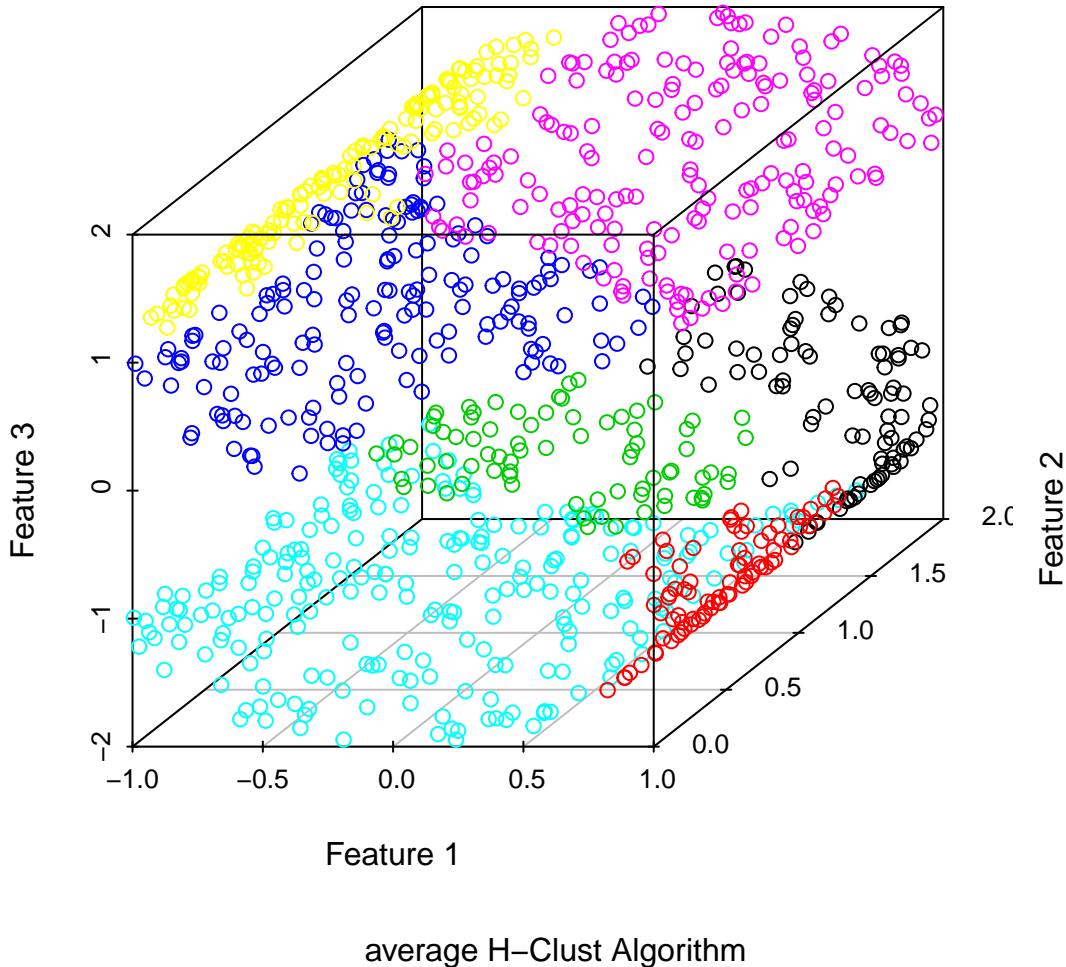
H-Clust Complete se comporta de manera similar a K-Medias tratando de buscar superficies cilíndricas que no están presentes en este conjunto de datos y por ello el desempeño es pobre.

H-Clust Average

Generando el modelo

```
model_hierarchical_average_s = eval_hclust_3D(
  distance = hierarchical_distance_s, mode = "average", centroids = 7,
  input = input_hierarchical_s, dataname = "s.csv")
```

Data set s.csv



Generando la matriz de confusión

```
table_model_hierarchical_average_s = order_confusion_matrix(  
  confusion_matrix(df_s$class, model_hierarchical_average_s))  
table_model_hierarchical_average_s
```

```
##      PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## TC1 146  0  0  0  0  0  0  
## TC2  43 100  0  0  0  0  0  
## TC3   0  46 114  0  0  0  0  
## TC4   0   0  61  50  0 23  0  
## TC5   0   0   0  28  55 54  0  
## TC6   0   0   0   0  26 16 76  
## TC7   0   0   0   0   0  0 162
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_average_s, nrow(df_s))
```

```
## Tasa de acierto: 64.300000
## Tasa de fallo: 35.700000
```

Conclusión general

Por la naturaleza del conjunto de datos, ninguno de los algoritmos evaluados se comporta de buena manera, ya que los algoritmos utilizados buscan superficies cilíndricas o tratan de ajustarse a formas extrañas si los puntos están muy unidos, como es el caso de **H-Clust Single**. Existen otros algoritmos de clustering que se adaptan mejor a este tipo de problemas, tales como **Spectral Clustering**.

help.csv

Cargando el conjunto de datos

```
df_help = read.csv("../data/help.csv", header = F)
```

Pre-procesamiento

Veamos el rango de número presentes en la columna de clases.

```
summary(df_help$V4)
```

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

Podemos ver cómo el rango va desde [-4.712, 14.14], queremos hacer que este rango empiece desde el número uno (1).

```
df_help$V5 = floor(df_help$V4) - min(floor(df_help$V4)) +1
```

Si volvemos a inspeccionar el rango de valores de las nuevas clases calculadas.

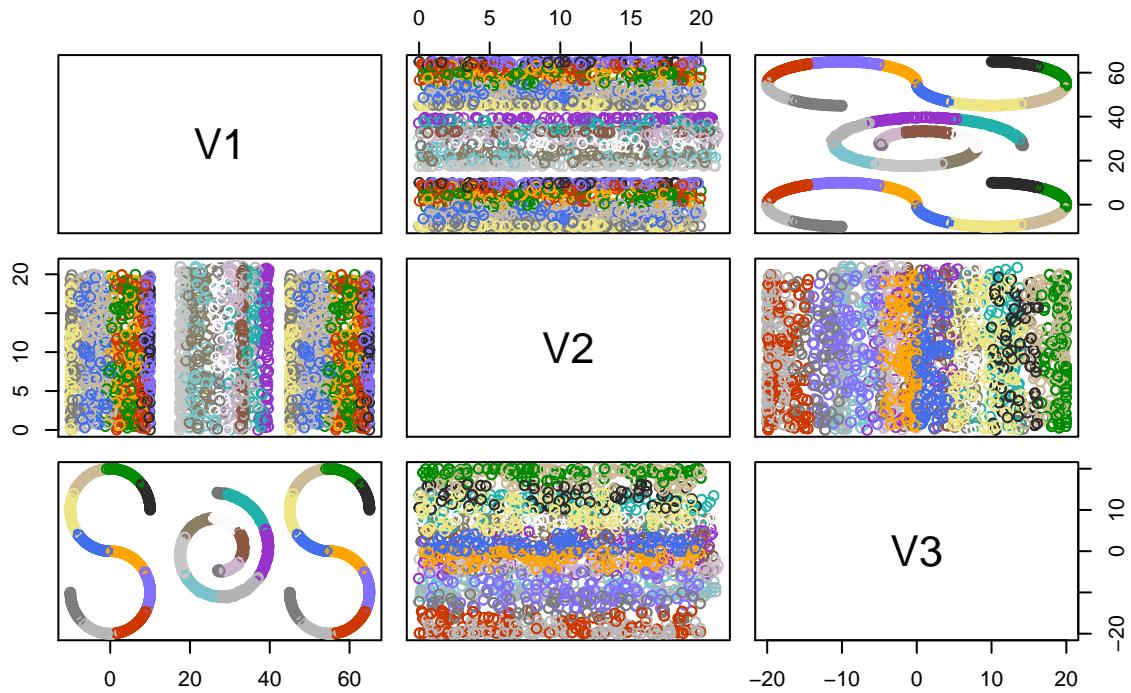
```
summary(df_help$V5)
```

```
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 1.000  4.000  8.000   8.637  13.000  20.000
```

Podemos ver cómo ahora el rango va desde [1, 20], mostrando el siguiente gráfico:

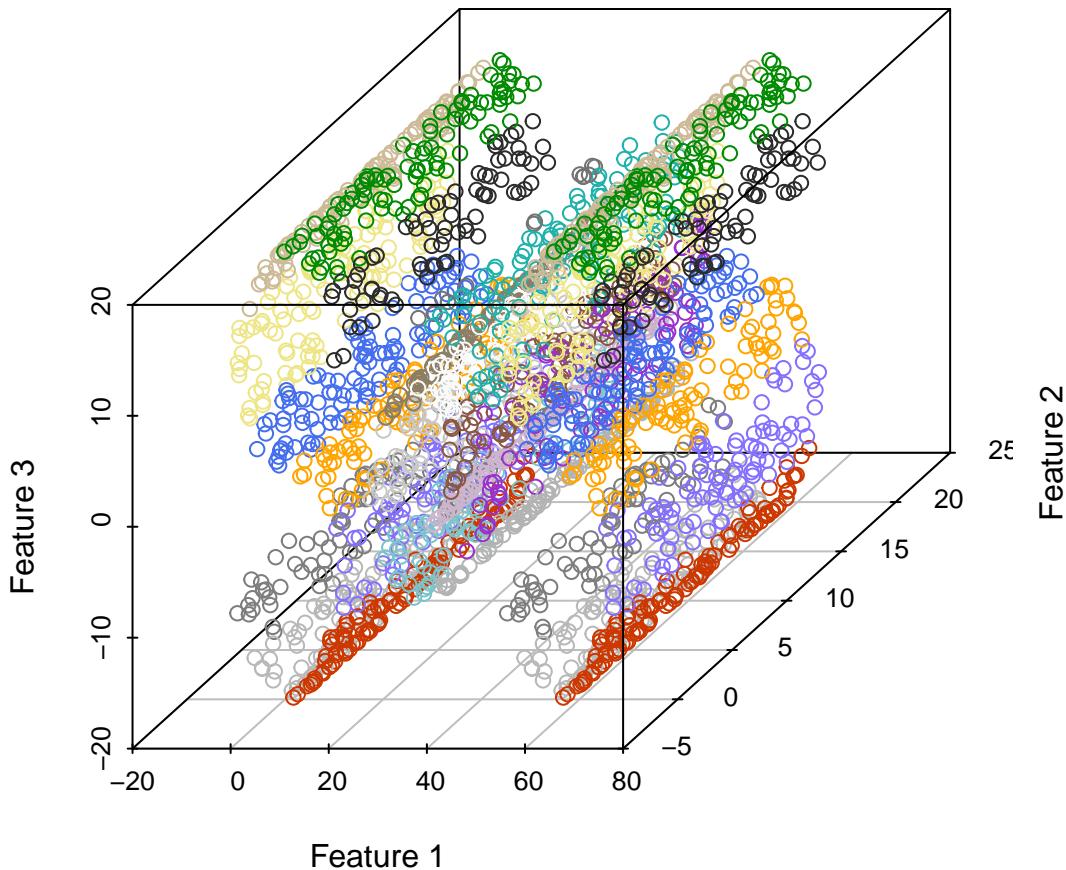
```
plot(df_help[,1:3], col= get_colors(20, df_help),
      main = "Matriz Dispersion help.csv")
```

Matrix Dispersion help.csv



```
scatterplot3d(df_help$V1, df_help$V2, df_help$V3, color = get_colors(20, df_help),
  type="p", xlab = "Feature 1", ylab = "Feature 2", zlab = "Feature 3",
  main= "Data Set help.csv")
```

Data Set help.csv



Donde podemos ver las letras “SOS”, las letras “S” tienen el mismo comportamiento que el conjunto de datos **s.csv** y la letra “O”, tiene el mismo comportamiento que nuestro conjunto de datos **h.csv** evaluados previamente. Dicho esto, y por el rendimiento y las conclusiones que sacamos luego de aplicar los diferentes algoritmos sobre cada uno de los elementos individualmente, donde pudimos notar que los algoritmos utilizados no se ajustaron precisamente bien a los modelos, podemos tratar este problema de una manera diferente.

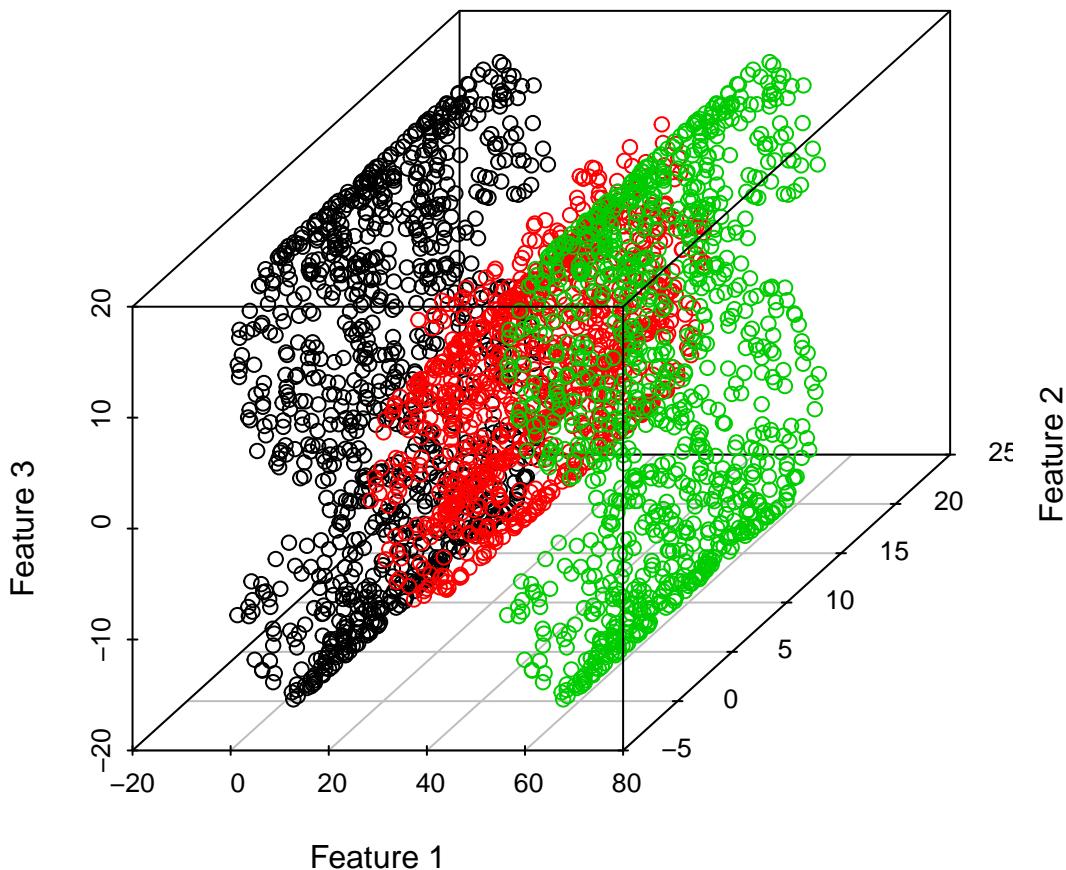
Al conjunto de datos presentar elementos que luego puden ser evaluados individualmente, se puede hacer una evaluación a dos (2) niveles, en donde en el primer nivel, se identificaría el tipo de letra que representa **S** u **O** y luego se podría aplicar un análisis detallado o específico a dicha letra luego de ser localizada, por lo mismo, nuestro criterio fue el de colocar sólo tres (3) clusters dentro de este conjunto de datos, de tal manera que luego que podamos identificar cada elemento individualmente podamos tratarlo de manera específica. Por lo tanto el primer paso, será el de llevar el rango de valor de las clases entre [1,3].

```
df_help$V5 = NULL  
df_help$class = 2  
df_help$class[df_help$V1 < 10] = 1  
df_help$class[df_help$V1 > 42] = 3
```

Lo que hicimos fue explorar sobre el eje “X” el dominio de las letras y lo delimitamos para poder marcar cada elemento de forma correcta, derivando en el siguiente gráfico.

```
scatterplot3d(df_help$V1, df_help$V2, df_help$V3, color = df_help$class,
  type="p", xlab = "Feature 1", ylab = "Feature 2", zlab = "Feature 3",
  main= "Data Set help.csv")
```

Data Set help.csv



Ahora que tenemos listo nuestro conjunto de datos en la forma en que queremos, podemos empezar a aplicar los distintos algoritmos.

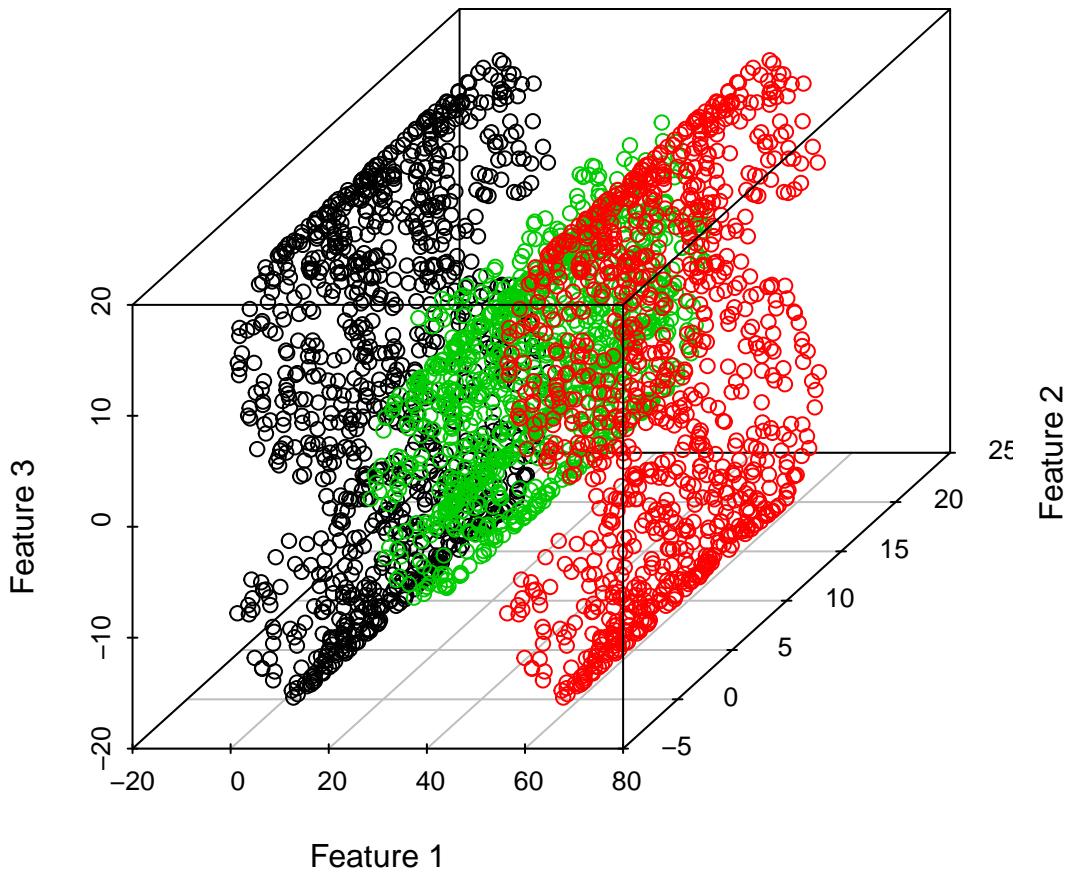
Aplicando los algoritmos

K-Medias

Generando el modelo

```
model_kmeans_help = eval_kmeans_3D(df = df_help, cstart = 1, cfinish = 3, k = 3, datafname = "help.csv")
```

Data Set help.csv



Generando la matriz de confusión

```
table_model_kmeans_help = order_confusion_matrix(confusion_matrix(df_help$class, model_kmeans_help$clus
table_model_kmeans_help
```

```
##      PC1  PC2  PC3
## TC1 999    0    0
## TC2   1 1000    0
## TC3   0    0 1000
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_kmeans_help, nrow(df_help))
```

```
## Tasa de acierto: 99.966667
## Tasa de fallo: 0.033333
```

Conclusión

El algoritmo presenta un excelente desempeño ya que puede ubicar las superficies cilíndricas “S” y esféricas “O” de gran manera, presentando una tasa de acierto excelente.

H-Clust

Necesitamos calcular la **matriz de distancias** antes de aplicar el algoritmo.

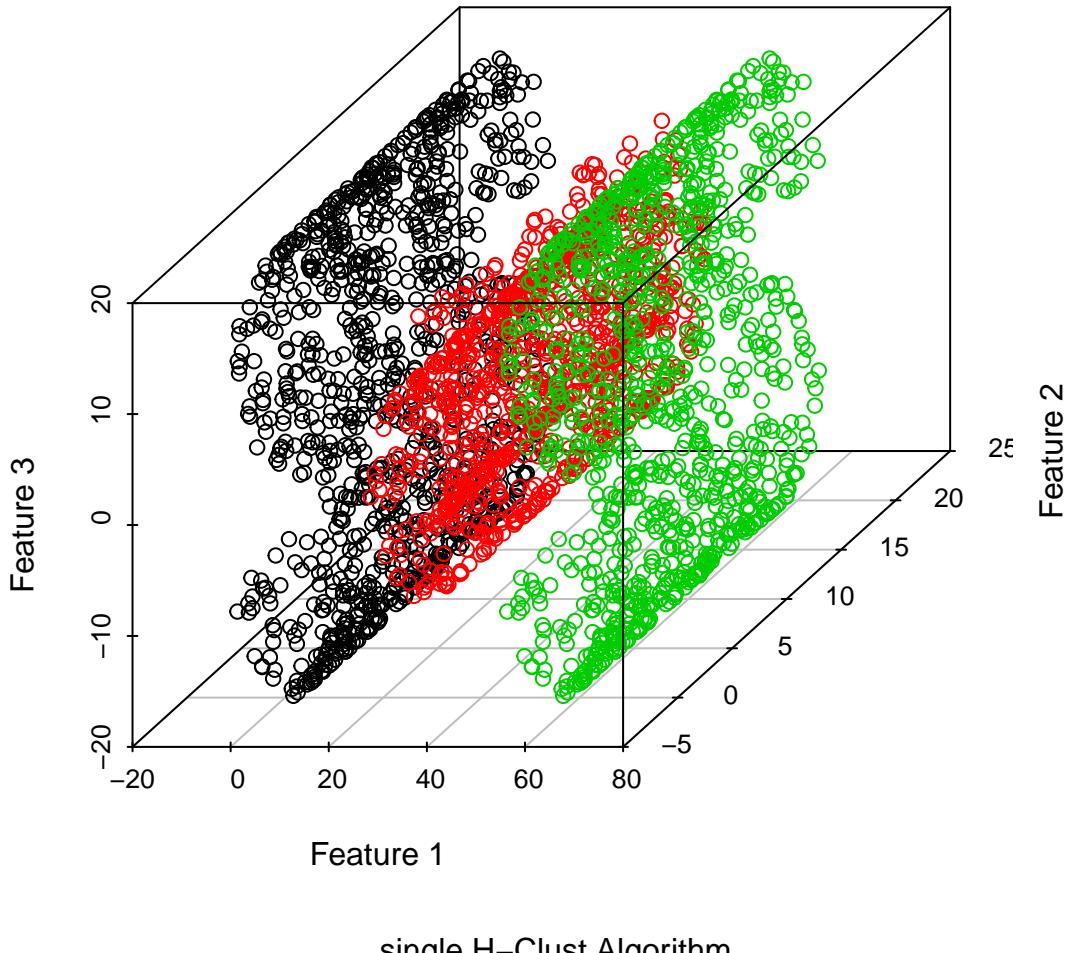
```
input_hierarchical_help = df_help
input_hierarchical_help$V4 = NULL
input_hierarchical_help$class = NULL
input_hierarchical_help = as.matrix(input_hierarchical_help)
hierarchical_distance_help = dist(input_hierarchical_help)
```

H-Clust Single

Generando el modelo

```
model_hierarchical_single_help = eval_hclust_3D(
  distance = hierarchical_distance_help, mode = "single",
  centroids = 3, input = input_hierarchical_help, dataname = "help.csv")
```

Data set help.csv



Generando la matriz de confusión

```
table_model_hierarchical_single_help = order_confusion_matrix(  
  confusion_matrix(df_help$class, model_hierarchical_single_help))  
table_model_hierarchical_single_help  
  
##      PC1   PC2   PC3  
## TC1 999     0     0  
## TC2   1 1000     0  
## TC3   0     0 1000
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_single_help, nrow(df_help))  
  
## Tasa de acierto: 99.966667  
## Tasa de fallo: 0.033333
```

Conclusión

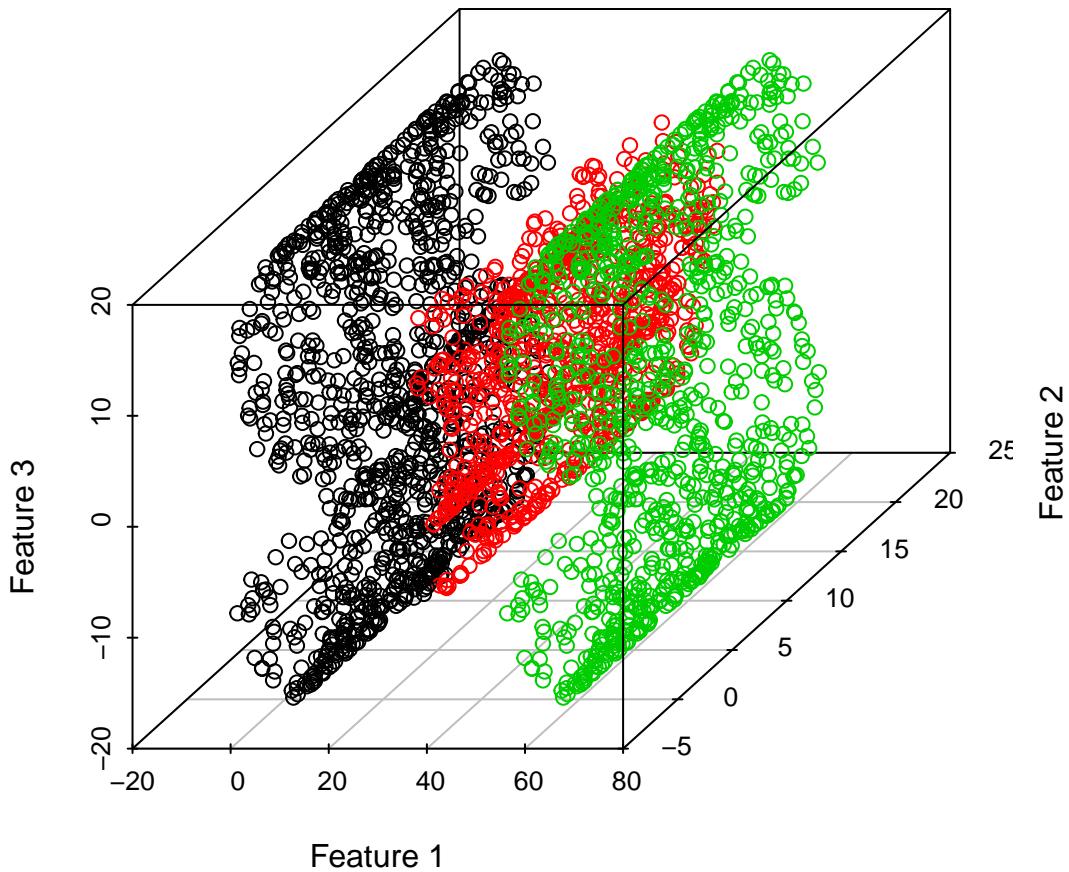
Presenta un excelente rendimiento por la estructura continua con fronteras bien separables entre sus elementos.

H-Clust Complete

Generando el modelo

```
model_hierarchical_complete_help = eval_hclust_3D(  
  distance = hierarchical_distance_help, mode = "complete",  
  centroids = 3, input = input_hierarchical_help, dataname = "help.csv")
```

Data set help.csv



complete H-Clust Algorithm

Generando la matriz de confusión

```
table_model_hierarchical_complete_help = order_confusion_matrix(  
  confusion_matrix(df_help$class, model_hierarchical_complete_help))  
table_model_hierarchical_complete_help
```

```
##      PC1 PC2 PC3  
## TC1  999   0   0  
## TC2  133 868   0  
## TC3    0  42 958
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_complete_help, nrow(df_help))
```

```
## Tasa de acierto: 94.166667  
## Tasa de fallo: 5.833333
```

Conclusión

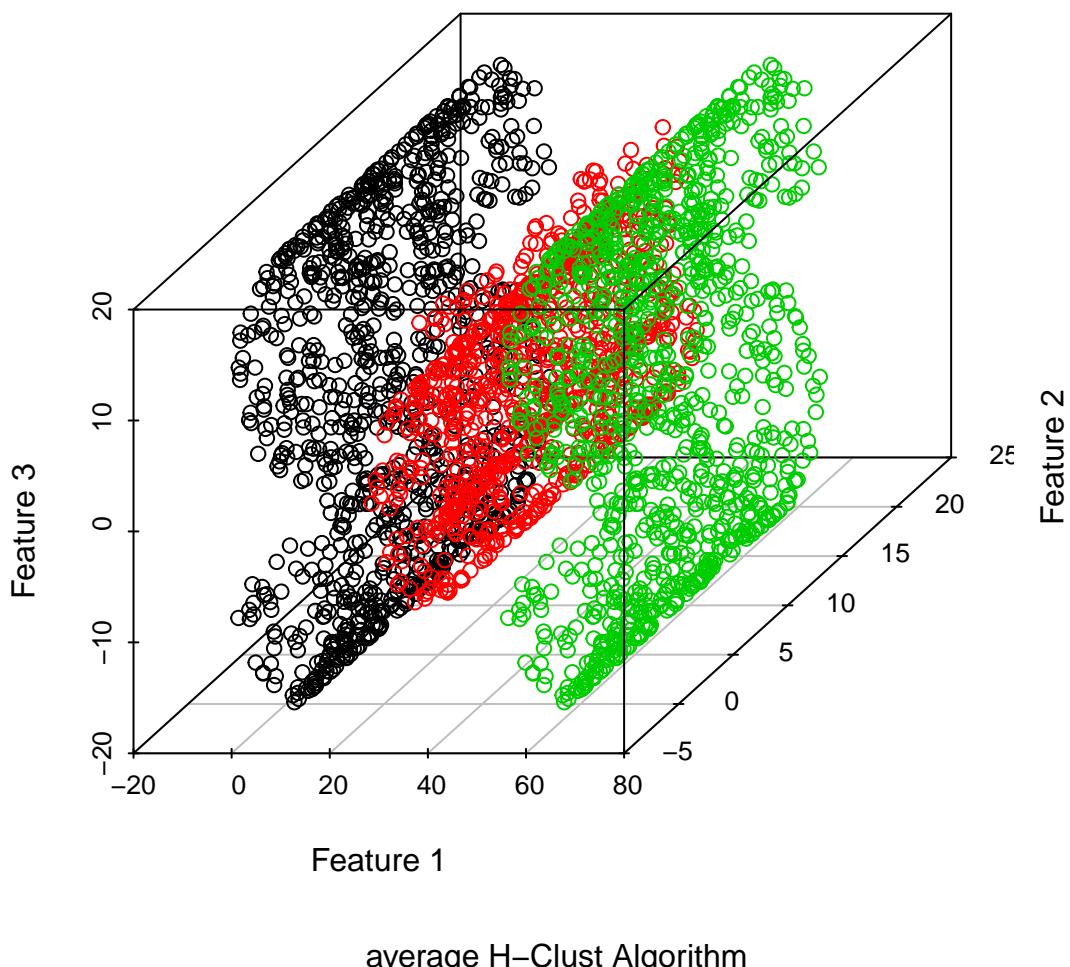
A pesar de que presenta una tasa alta de aciertos, no funciona tan bien como los otros dos algoritmos evaluados anteriormente, ya que mezcla parte de las “S” con la “O”.

H-Clust Average

Generando el modelo

```
model_hierarchical_average_help = eval_hclust_3D(  
    distance = hierarchical_distance_help, mode = "average",  
    centroids = 3, input = input_hierarchical_help, dataname = "help.csv")
```

Data set help.csv



Generando la matriz de confusión

```
table_model_hierarchical_average_help = order_confusion_matrix(  
  confusion_matrix(df_help$class, model_hierarchical_average_help))  
table_model_hierarchical_average_help
```

```
##      PC1  PC2  PC3  
## TC1 999    0    0  
## TC2   1 944   56  
## TC3   0    0 1000
```

Evaluando la matriz de confusión

```
confusion_matrix_evaluation(table_model_hierarchical_average_help, nrow(df_help))  
  
## Tasa de acierto: 98.100000  
## Tasa de fallo: 1.900000
```

Conclusión

Presenta una alta tasa de aciertos, sólo confundiendo pocos elementos debido a que la frontera entre los dos elementos es muy corta, sin embargo, presenta un desempeño más que bueno.

Conclusión general

El algoritmo con mejor desempeño fue **H-Clust Single**, que sólo falló en un elemento a la hora de clasificar, sin embargo, en promedio todos los algoritmos se comportaron de una forma excelente. Con este enfoque de dos capas, tenemos un gran modelo para identificar un elemento y luego, aplicamos otro modelo individualmente a cada elemnto para poder clusterizar individualmente.

Este enfoque a dos niveles se puede presentar en escenarios cómo en ataques de seguridad o detección de anomalías, en los cuales, primero se aplica un modelo para detectar una anomalía y luego, se aplica otro modelo para determinar que tipo de anomalía es.

Análisis sin columna de clase

Consta de un conjunto de datos llamado **guess.csv**, este conjunto de datos no tiene columna de clase, de tal manera que no podremos hacer evaluaciones del modelo desde el punto de vista de una matriz de confusión. Este tipo de escenario donde no se puede evaluar el modelo ya que no se tiene a priori la clase a la cuál pertenece cada elemento del conjunto, además resalta la necesidad de hacer un buen análisis exploratorio a fin de elegir la cantidad de clases ideal a pasar como parámetro a los diferentes algoritmos.

guess.csv

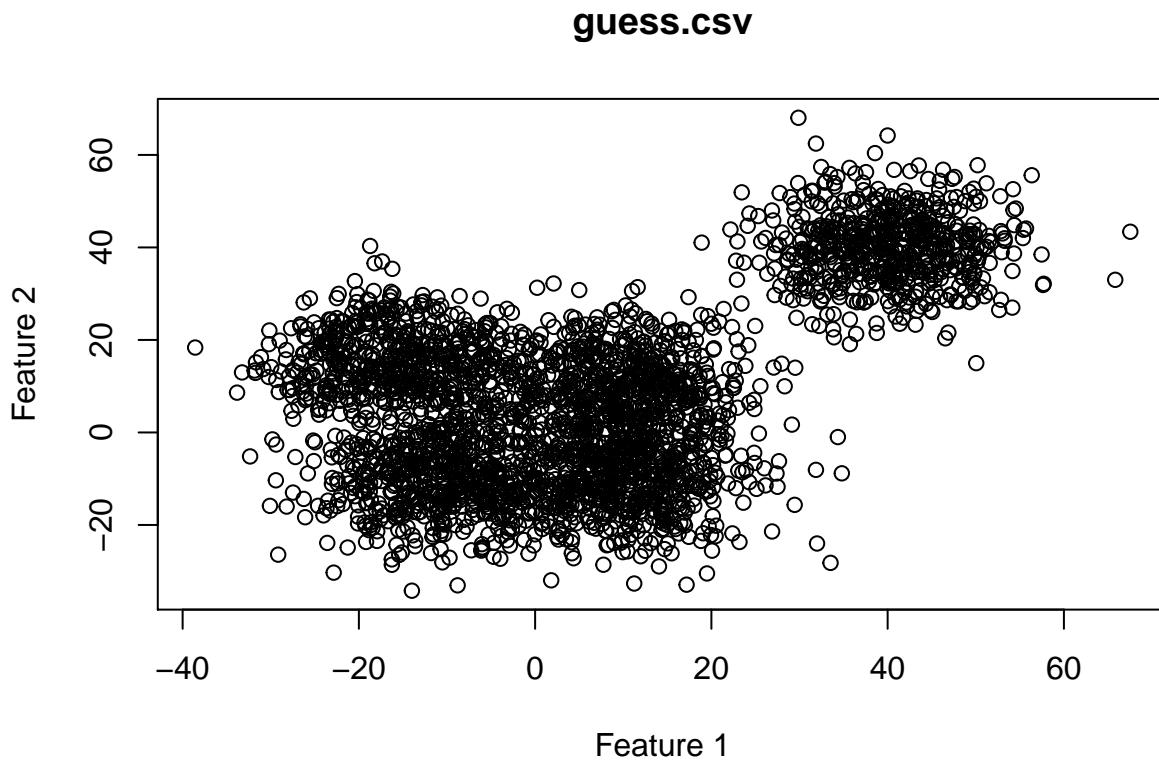
Cargando el conjunto de datos

```
df_guess = read.csv("../data/guess.csv", header = F)
```

Análisis exploratorio

Veamos gráficamente al conjunto de datos **guess.csv**.

```
plot(df_guess, xlab = "Feature 1", ylab = "Feature 2",
     main = "guess.csv")
```



Podemos observar cómo a simple vista existen al menos dos (2) clusters bien separados (Parte inferior izquierda vs parte superior derecha), sin embargo el cluster más grande podría parecer que tiene alguna otra separación que a simple vista no se puede identificar. Dicho esto surge la siguiente pregunta: **¿Cómo saber el número de clusters a elegir?**

Bueno existen varias técnicas, la que aplicaremos en este caso es la llamada **Codo de Jambu** en la cuál estaremos probando con diferentes valores para **k** centroides hasta elegir aquel que nos proporcione el modelo con menor error.

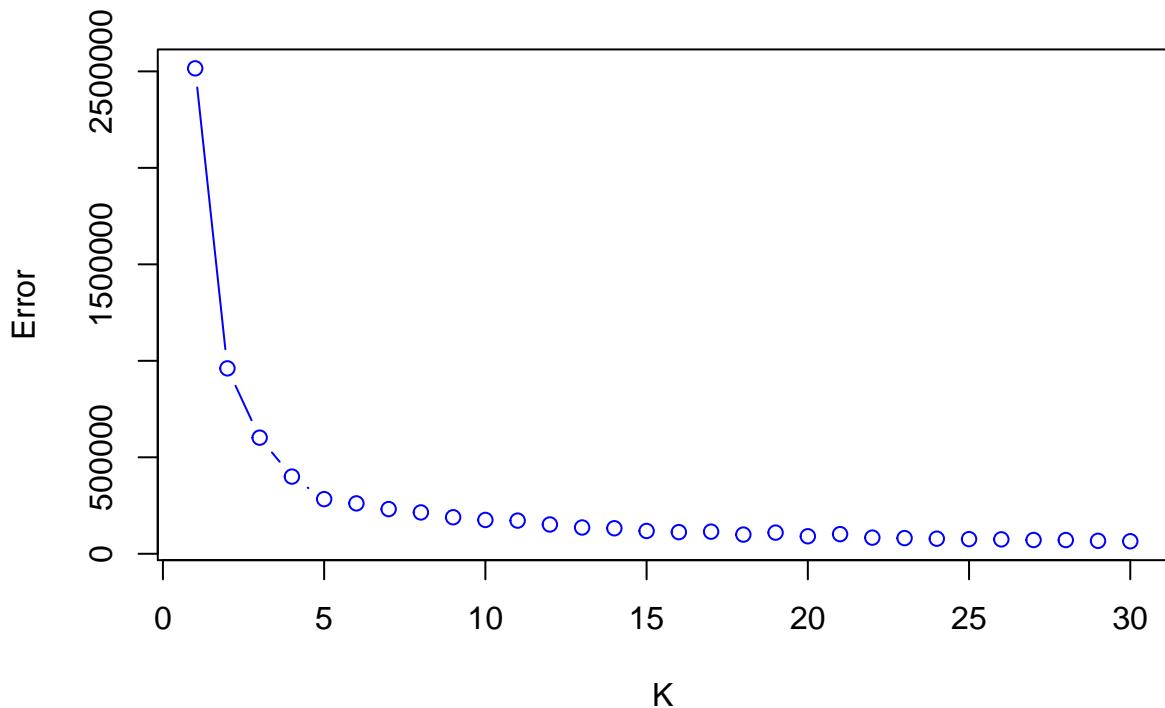
El método de **Codo de Jambu** es un método de calibración que lo que hace es tratar de minimizar la suma de los errores para cada iteración de **k** centroides.

```
valor = rep(0,30)

for (k in 1:30)
{
  model_kmeans_guess_jambu = kmeans(df_guess, k)
  valor[k] = model_kmeans_guess_jambu$tot.withinss
}

plot(valor, col = "blue", type = "b",
      xlab = "K", ylab = "Error",
      main = "Codo de Jambu")
```

Codo de Jambu



Como se puede ver en la gráfica, la suma de los errores converge a cero (0) a medida que tendemos al infinito o a un número alto de centroides. Sin embargo la interpretación de este gráfico es la de elegir aquel **K** en el cuál la curva empieza a suavizarse y variar de manera visualmente nula, que en este caso es **K = 5**.

Dicho esto, empezaremos a evaluar nuestros algoritmos con un número de centroides igual a cinco (5).

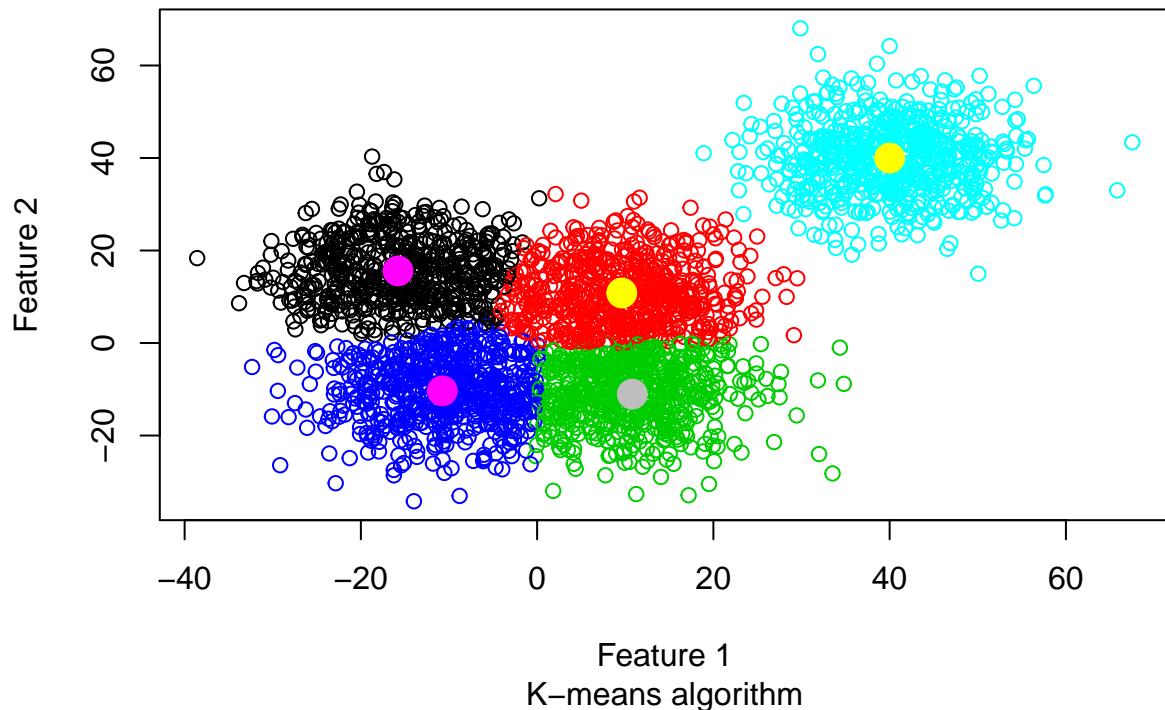
Aplicando los algoritmos

K-Medias

Generando el modelo

```
model_kmeans_guess = eval_kmeans(df = df_guess, cstart = 1, cfinish = 2, k = 5, dataname = "guess.csv")
```

Data set guess.csv



Conclusión

En el gráfico podemos ver que los cinco (5) clusters generados por el modelo tienen un comportamiento circular bien definido, lo que nos hace pensar que el número de clases fue bien seleccionado y que de esta manera hicimos una buena elección del número de centroides con el método de calibración de modelos de **Codo de Jambu**.

H-Clust

Necesitamos calcular la **matriz de distancias** antes de aplicar el algoritmo.

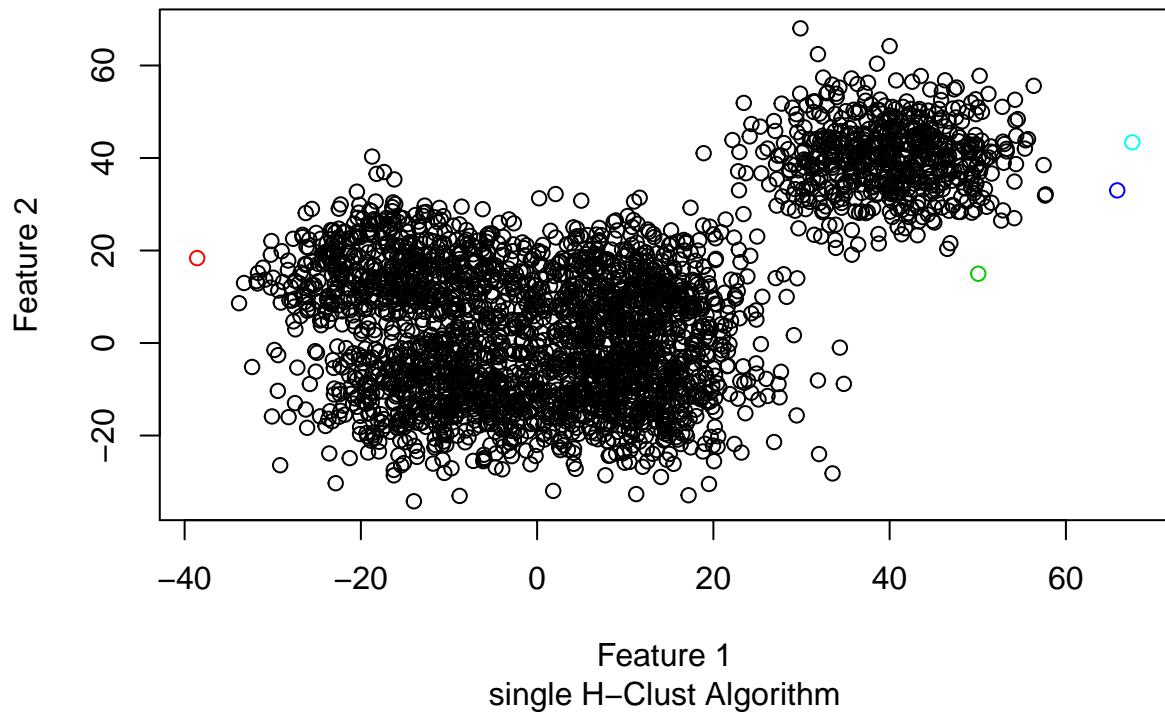
```
input_hierarchical_guess = df_guess
input_hierarchical_guess$V3 = NULL
input_hierarchical_guess = as.matrix(input_hierarchical_guess)
hierarchical_distance_guess = dist(input_hierarchical_guess)
```

H-Clust Single

Generando el modelo

```
model_hierarchical_single_guess = eval_hclust(
  distance = hierarchical_distance_guess, mode = "single",
  centroids = 5, input = input_hierarchical_guess, dataname = "guess.csv")
```

Data set guess.csv



Conclusión

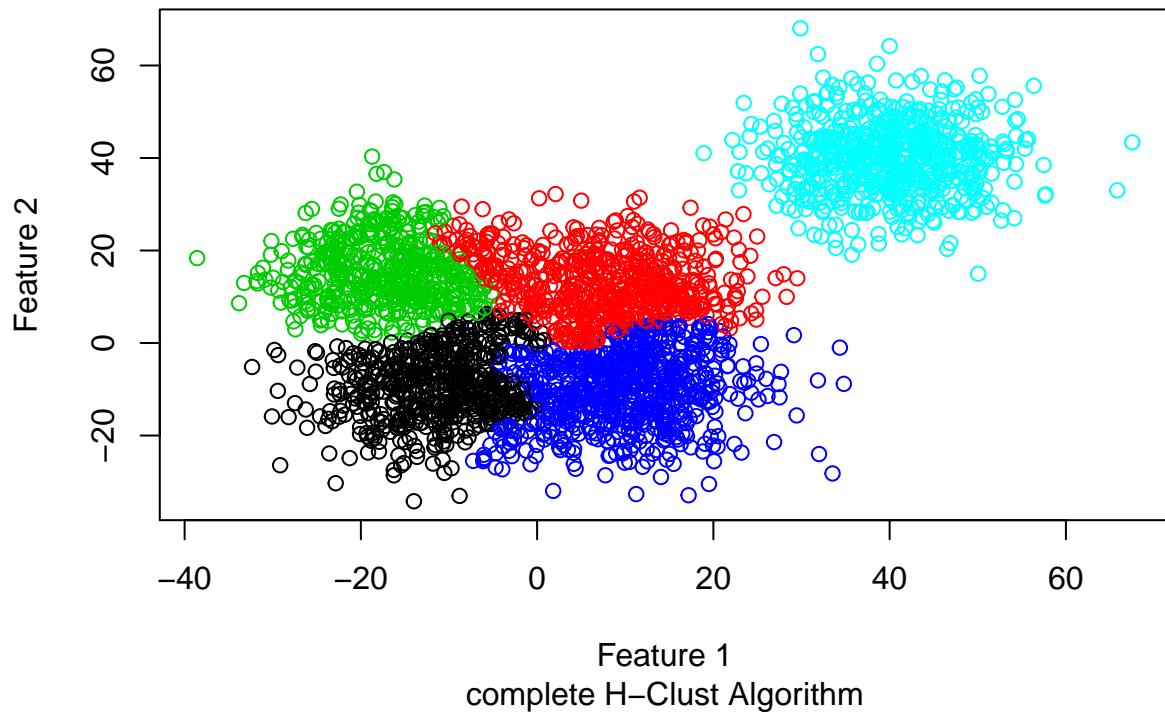
Cómo hemos visto a lo largo del informe, este algoritmo no se ajusta bien a este tipo de estructuras donde el conjunto de datos presenta formas circulares y no tiene fronteras bien marcadas, y básicamente lo que hace es colocar todos los elementos dentro de un mismo cluster.

H-Clust Complete

Generando el modelo

```
model_hierarchical_complete_guess = eval_hclust(  
    distance = hierarchical_distance_guess, mode = "complete",  
    centroids = 5, input = input_hierarchical_guess, dataname = "guess.csv")
```

Data set guess.csv



Conclusión

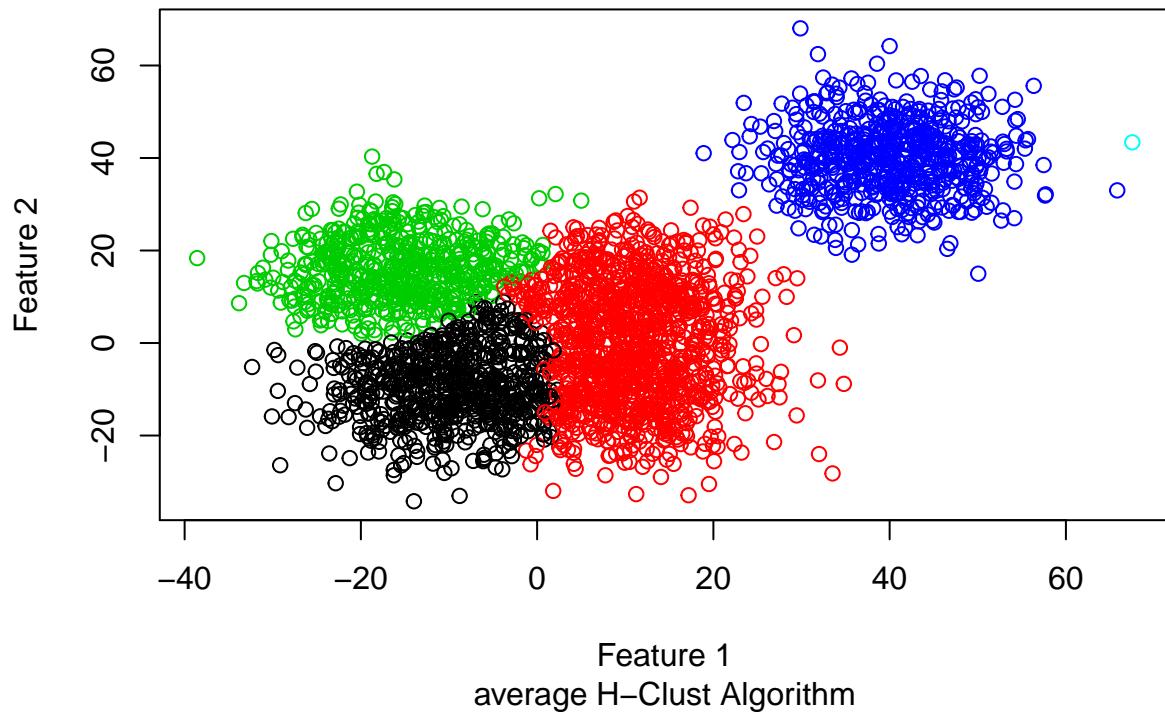
Se comporta parecido a K-Medias, sin embargo los clusters no son muy redondos, cosa que puede ser buena o mala dependiendo de la naturaleza real de los clusters en la vida real.

H-Clust Average

Generando el modelo

```
model_hierarchical_average_guess = eval_hclust(  
    distance = hierarchical_distance_guess, mode = "average",  
    centroids = 5, input = input_hierarchical_guess, dataname = "guess.csv")
```

Data set guess.csv



Conclusión

Vemos cómo el comportamiento no tiene mucho sentido debido a que uno de los clusters parece ocupar el lugar de dos (2) simultáneamente dejando a un cluster sólo con un elemento.

Conclusión general

El algoritmo que parece ajustarse mejor a este conjunto de datos es **K-Medias**, ya hemos visto que la naturaleza del conjunto de datos circular es perfecta para este algoritmo. Es importante destacar la funcionalidad del método de calibración del **Codo de Jambu** debido a que es un algoritmo sumamente simple pero muy eficaz para la selección de los centroides.