

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 de 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 el la técnica de aprendizaje no supervisada.

Dependencias

Utilizaremos la biblioteca “rgl” 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 instalar 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(rgl)
```

Implementación

Sólo utilizaremos la biblioteca **rgl** 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)
  plot3d(x = df$V1, y = df$V2, z = df$V3, type = "s" ,col = model_kmeans$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 general 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 general 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)
  plot3d(x = input[,1], y = input[,2], z = input[,3], type = "s",
    main = paste(c("Data set", dataname), collapse = " "),
    sub = paste(c(mode,"H-Clust Algorithm"), collapse = " "),
    xlab = "Feature 1", ylab = "Feature 2", zlab = "Feature 3",
    col = 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:nrow(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(auxi[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)
}
```

Análisis con clases enteras

Consta de tres conjuntos de datos:

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

Vamos a empezar a trabajar en ellos

a.csv

Cargando el conjunto de datos

Cargamos en df_a el conjunto de datos

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

Pre-procesamiento

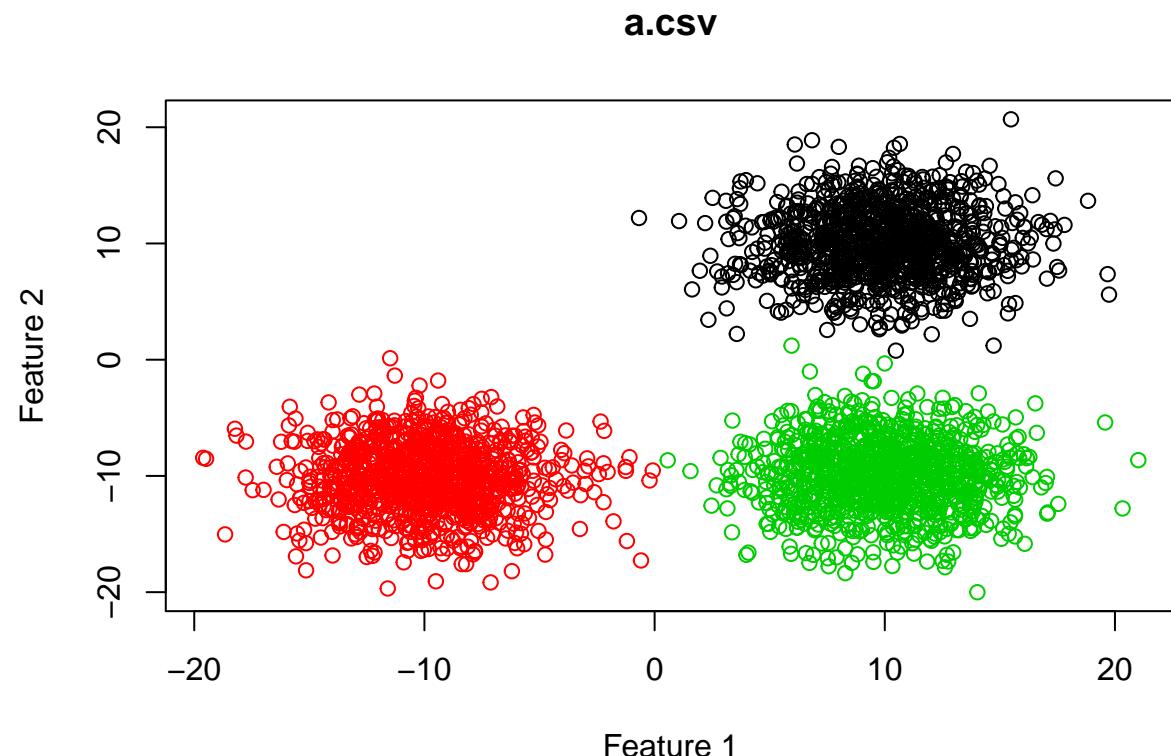
La columna de clases está en el rango [0,1] y la llevaremos al rango [1,2]

```
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.

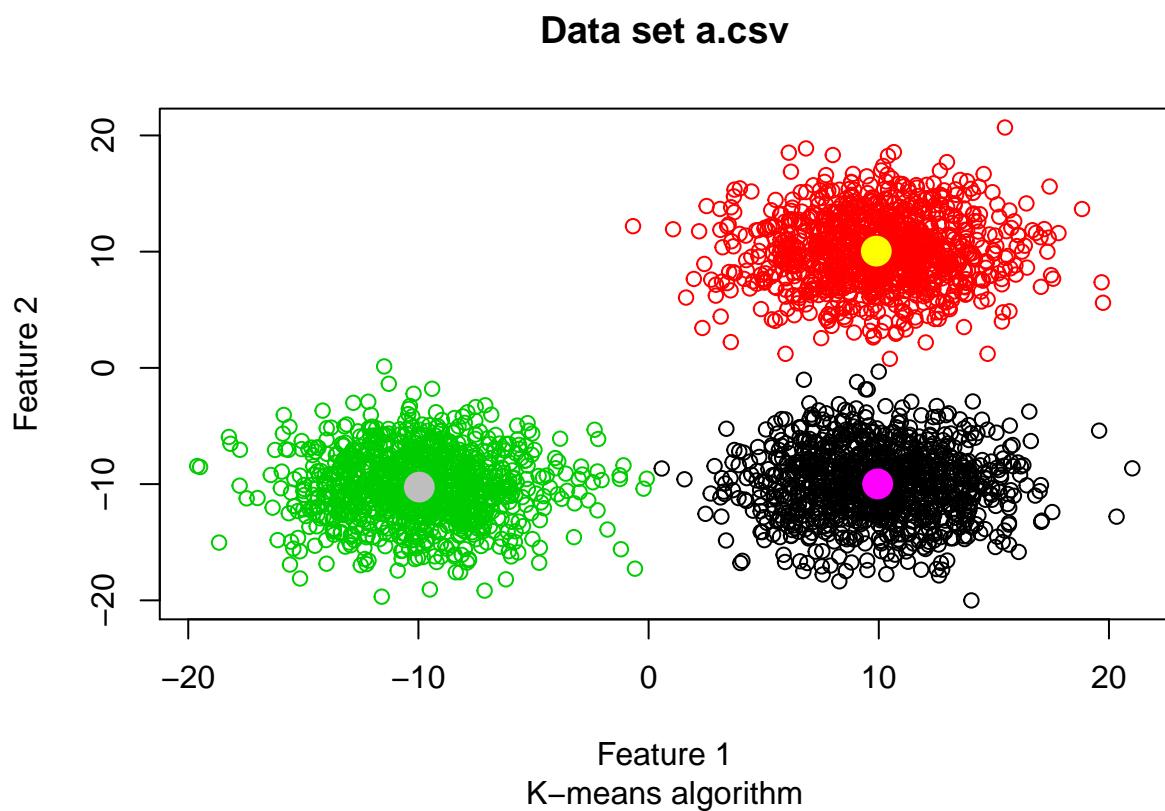
Aplicando los algoritmos

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 o 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.

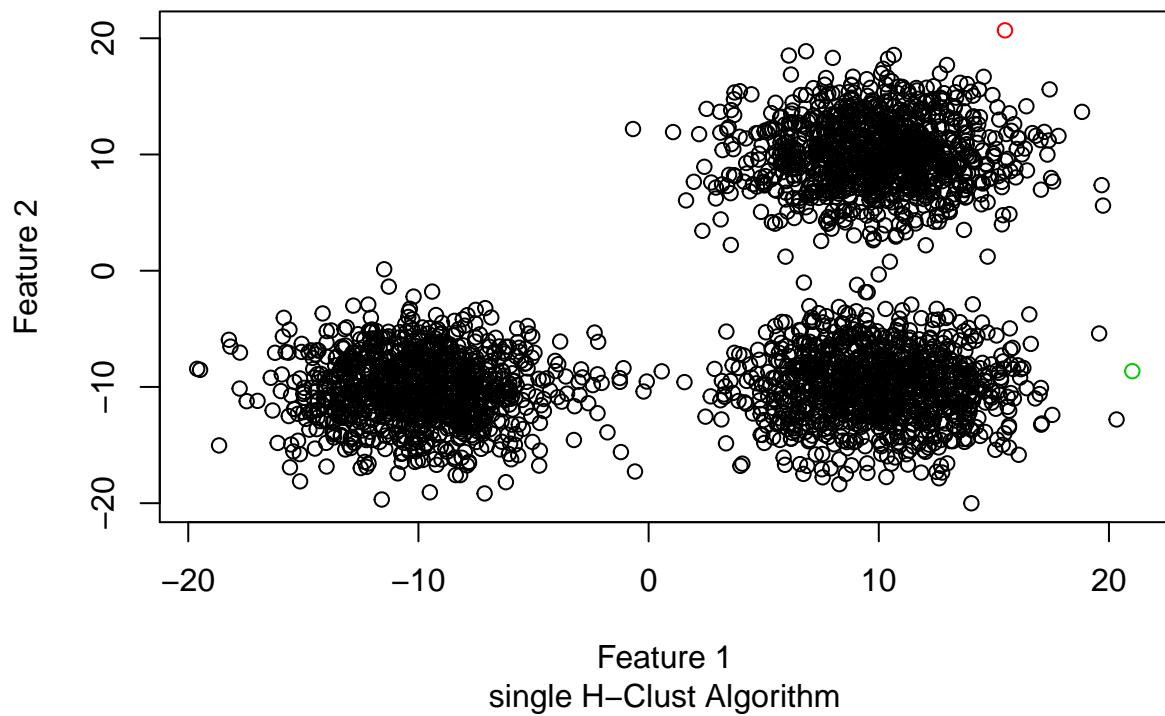
```
input_hierarchical_a = df_a
input_hierarchical_a$V3 = NULL
input_hierarchical_a = as.matrix(input_hierarchical_a)
hierarchical_distance_a = dist(input_hierarchical_a)
```

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



Feature 1
single H-Clust Algorithm

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

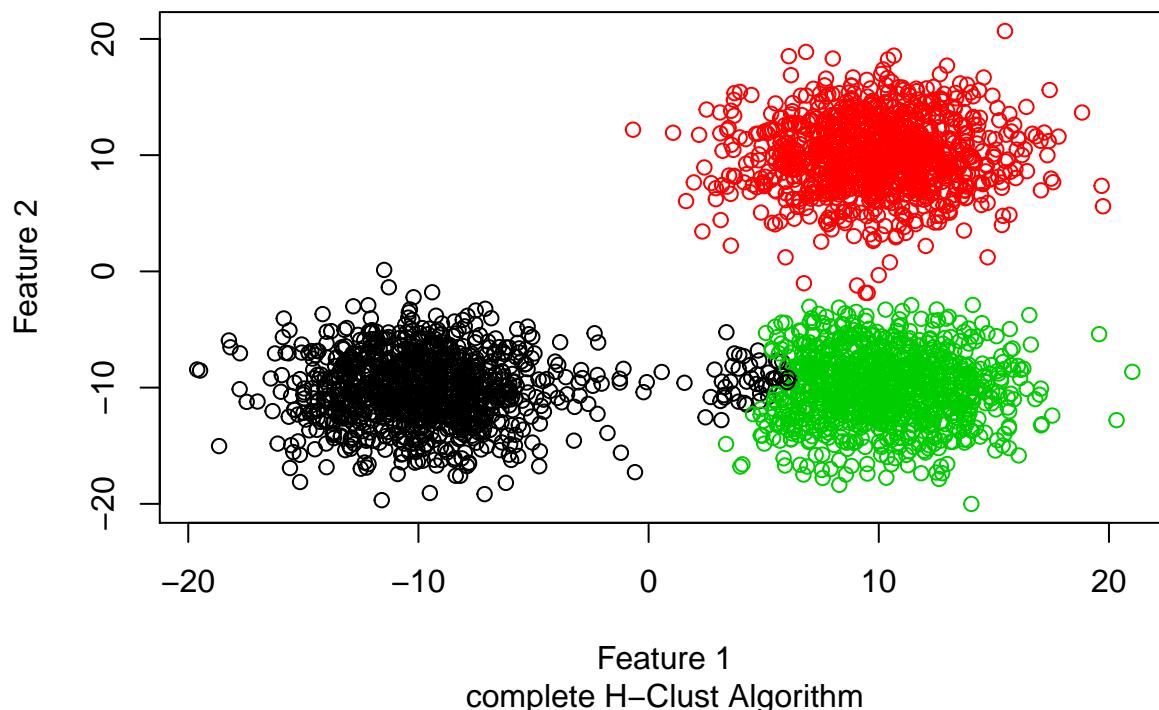
H-Clust single no muestra un gran rendimiento debido a que el algoritmo no está diseñado para buscar figuras circulares o cilíndricas, sino figuras no circulares.

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



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

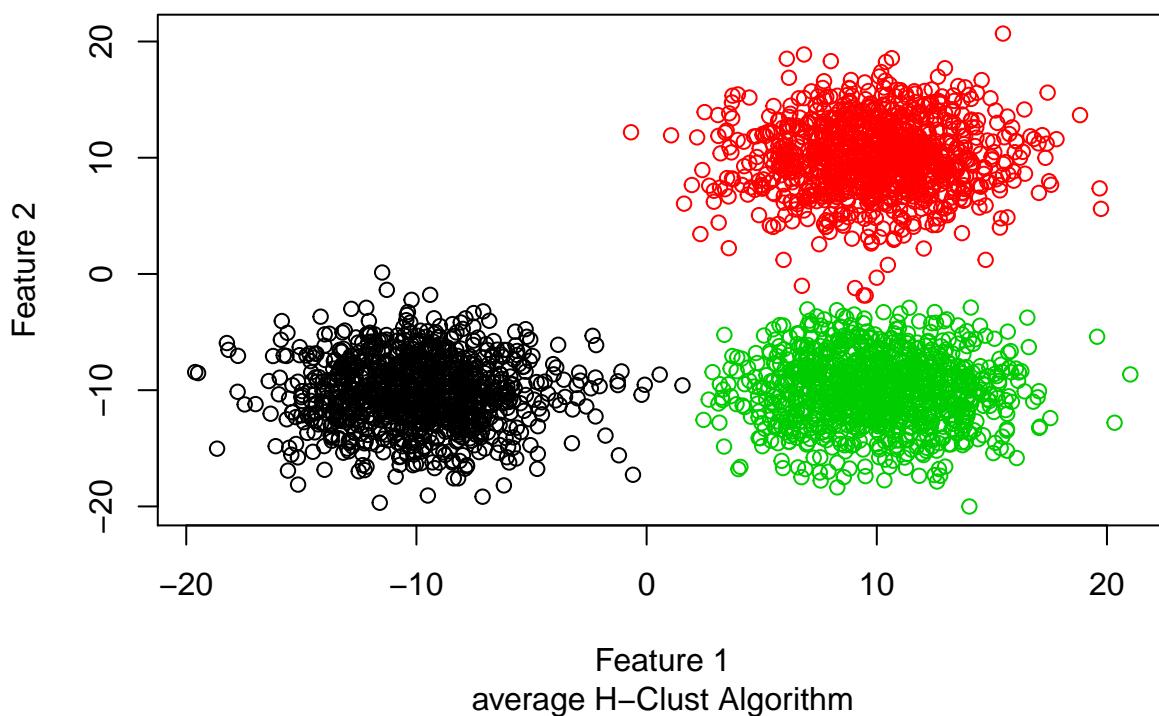
H-Clust complete se comporta parecido a K-Medias con respecto a que trata de buscar figuras circulares o cilíndricas, sin embargo en este caso no funcionó de la mejor manera, ya que confunde parte de los elementos de un cluster notablemente, disminuyendo así la tasa de aciertos.

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")
```

Data set 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

Cargamos en **df_a_big** 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)
```

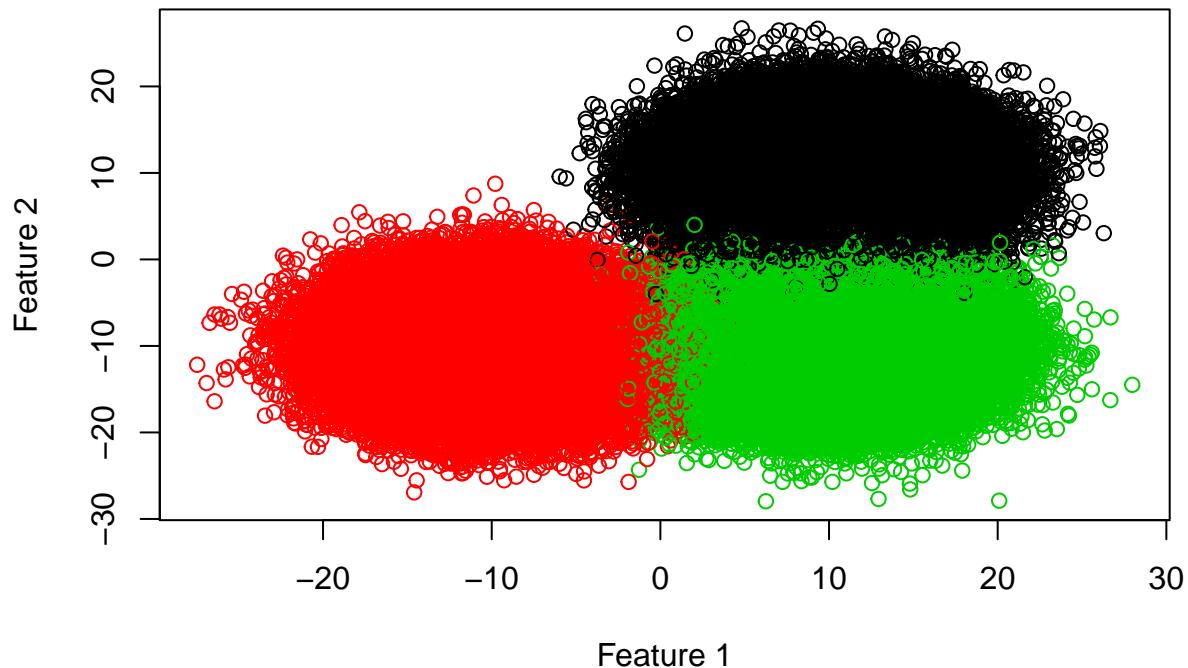
```
## [1] 300000
```

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 el 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.

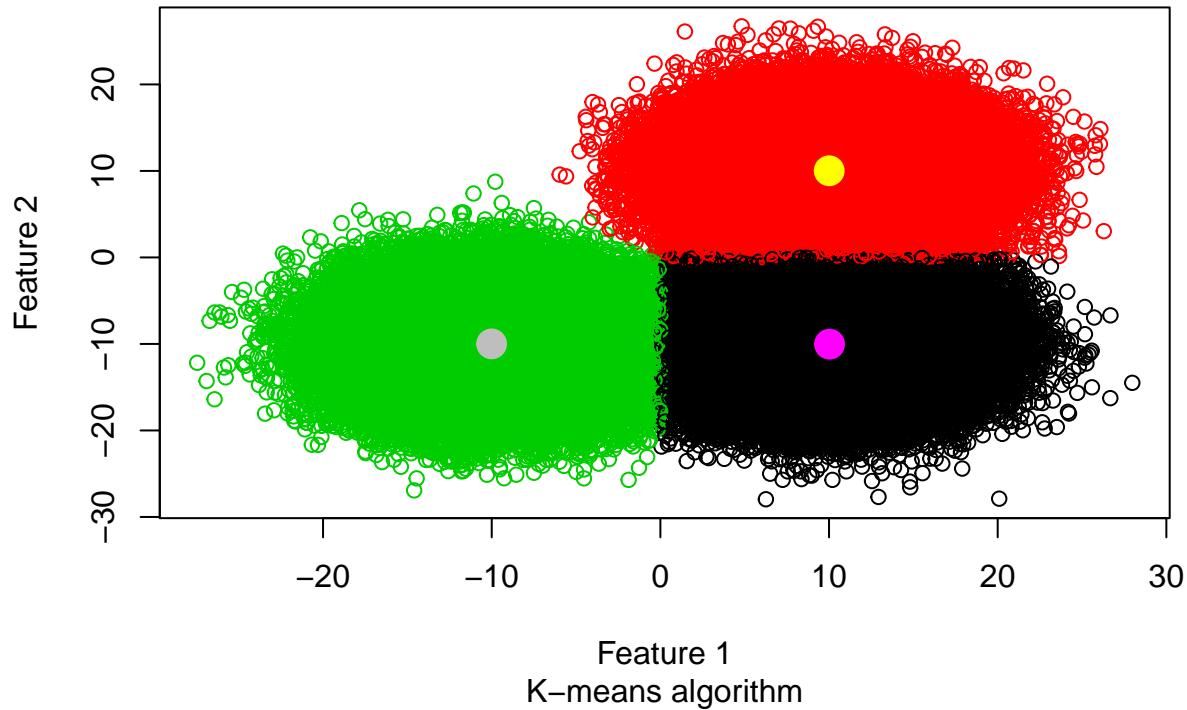
Aplicando los algoritmos

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.

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$V3, model_kmeans_a_big$clus
```

```
##          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.