codigoTFG

Jorge de AndrÃƒÆ’Ã†â€™Ãƒâ€šÃ‚Â©s

12 de enero de 2019

Lo primero que tengo que hacer es importar el dataset que he creado

dataset <- read.csv("C:/Users/jorge/Desktop/Documentos Clase/Universidad/4ºCarrera/1er Cuatrimestre/Inteligencia Artificial/Trabajo Fin de Asignatura/datos.txt", header = TRUE)

Ahora lo que hago es pasarlo a una matriz, quitando tanto el nombre (que no me interesa) como la etiqueta (que no la necesito por ahora)

matriz.pacientes.etiquetas <- dataset[, -1]  
matriz.pacientes.datos <- matriz.pacientes.etiquetas[, -25]

# Análisis Exploratorio

Primero compruebo que todos los datos tienen un tipo correcto.

sapply(matriz.pacientes.datos, class)

## edad sex rel\_ctxo\_rel\_mala rel\_ctxo\_trauma   
## "integer" "integer" "integer" "integer"   
## rel\_ctxo\_buena ed\_perm ed\_norm ed\_estr   
## "integer" "integer" "integer" "integer"   
## resil\_ba resil\_me resil\_al pen\_dic   
## "integer" "integer" "integer" "integer"   
## gen\_ex etiq fil\_men max\_min   
## "integer" "integer" "integer" "integer"   
## conc\_arb pseu\_res deb raz\_emo   
## "integer" "integer" "integer" "integer"   
## inhib asert agres impuls   
## "integer" "integer" "integer" "integer"

Veo la media de la edad de los pacientes y el rango en el que se mueve

mean(matriz.pacientes.datos[, 1])

## [1] 26.46269

range(matriz.pacientes.datos[, 1])

## [1] 13 52

Finalmente, veo un resúmen de cada columna

summary(matriz.pacientes.datos)

## edad sex rel\_ctxo\_rel\_mala rel\_ctxo\_trauma   
## Min. :13.00 Min. :0.000 Min. :0.0000 Min. :0.0000   
## 1st Qu.:19.50 1st Qu.:0.000 1st Qu.:0.0000 1st Qu.:0.0000   
## Median :25.00 Median :0.000 Median :0.0000 Median :0.0000   
## Mean :26.46 Mean :0.209 Mean :0.1343 Mean :0.3582   
## 3rd Qu.:30.50 3rd Qu.:0.000 3rd Qu.:0.0000 3rd Qu.:1.0000   
## Max. :52.00 Max. :1.000 Max. :1.0000 Max. :1.0000   
## rel\_ctxo\_buena ed\_perm ed\_norm ed\_estr   
## Min. :0.0000 Min. :0.0000 Min. :0.0000 Min. :0.0000   
## 1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.:0.0000   
## Median :1.0000 Median :0.0000 Median :0.0000 Median :0.0000   
## Mean :0.5075 Mean :0.2836 Mean :0.4925 Mean :0.2239   
## 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.:0.0000   
## Max. :1.0000 Max. :1.0000 Max. :1.0000 Max. :1.0000   
## resil\_ba resil\_me resil\_al pen\_dic   
## Min. :0.0000 Min. :0.0000 Min. :0.00000 Min. :0.0000   
## 1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.:0.00000 1st Qu.:1.0000   
## Median :1.0000 Median :0.0000 Median :0.00000 Median :1.0000   
## Mean :0.5672 Mean :0.4179 Mean :0.01493 Mean :0.8955   
## 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.:0.00000 3rd Qu.:1.0000   
## Max. :1.0000 Max. :1.0000 Max. :1.00000 Max. :1.0000   
## gen\_ex etiq fil\_men max\_min   
## Min. :0.0000 Min. :0.0000 Min. :0.000 Min. :0.0000   
## 1st Qu.:1.0000 1st Qu.:0.5000 1st Qu.:1.000 1st Qu.:1.0000   
## Median :1.0000 Median :1.0000 Median :1.000 Median :1.0000   
## Mean :0.9552 Mean :0.7463 Mean :0.791 Mean :0.9701   
## 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.:1.000 3rd Qu.:1.0000   
## Max. :1.0000 Max. :1.0000 Max. :1.000 Max. :1.0000   
## conc\_arb pseu\_res deb raz\_emo   
## Min. :0.0000 Min. :0.0000 Min. :0.0000 Min. :0.000   
## 1st Qu.:1.0000 1st Qu.:0.0000 1st Qu.:1.0000 1st Qu.:1.000   
## Median :1.0000 Median :1.0000 Median :1.0000 Median :1.000   
## Mean :0.9851 Mean :0.5075 Mean :0.9403 Mean :0.791   
## 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.:1.000   
## Max. :1.0000 Max. :1.0000 Max. :1.0000 Max. :1.000   
## inhib asert agres impuls   
## Min. :0.0000 Min. :0.0000 Min. :0.000 Min. :0.0000   
## 1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.:0.000 1st Qu.:0.0000   
## Median :1.0000 Median :0.0000 Median :0.000 Median :1.0000   
## Mean :0.6567 Mean :0.1343 Mean :0.209 Mean :0.6119   
## 3rd Qu.:1.0000 3rd Qu.:0.0000 3rd Qu.:0.000 3rd Qu.:1.0000   
## Max. :1.0000 Max. :1.0000 Max. :1.000 Max. :1.0000

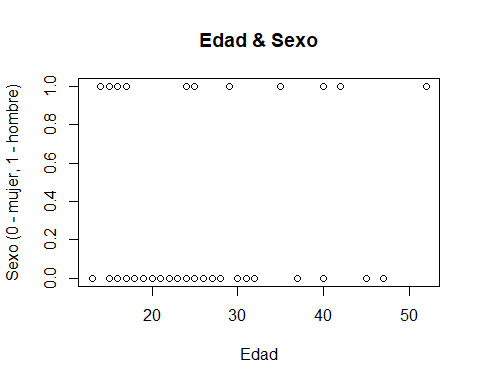
Como se puede ver, los datos de los pacientes están muy distanciados, y además su media es muy alta. Así, la media de la edad difiere enormemente del resto de valores de la matriz. Debido a ello, debemos de hacer un preprocesado de los datos del problema.

Antes que este preprocesado,voy a hacer la visualización de algunas relaciones entre variables, de tal manera que podamos ver gráficamente algunos aspectos interesantes:

## Visualización de Datos

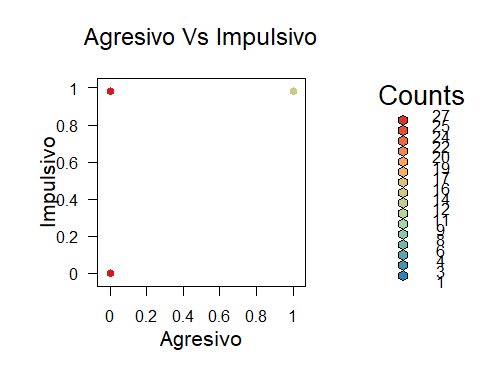
Ahora voy a sacar un plot para ver la relación entre la edad y el sexo de las personas que están en consulta

plot(matriz.pacientes.datos[,1], matriz.pacientes.datos[,2], xlab="Edad", ylab="Sexo (0 - mujer, 1 - hombre)", main="Edad & Sexo");



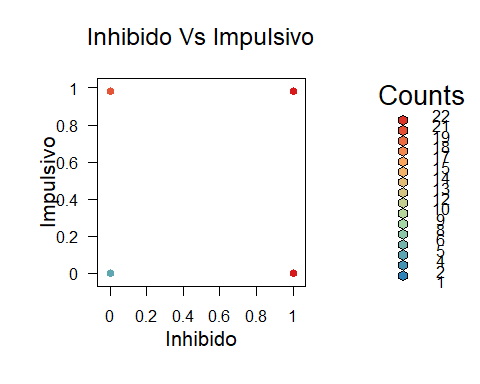
Otro plot para ver la correlación entre ser agresivo y ser impulsivo

#install.packages("hexbin")  
#install.packages("RColorBrewer")  
  
library(hexbin)  
library(RColorBrewer)  
  
rf <- colorRampPalette(rev(brewer.pal(4,'Spectral')))  
df <- data.frame(matriz.pacientes.datos[, 23], matriz.pacientes.datos[, 24])  
h <- hexbin(df)  
plot(h, colramp=rf, xlab="Agresivo", ylab="Impulsivo", main="Agresivo Vs Impulsivo")



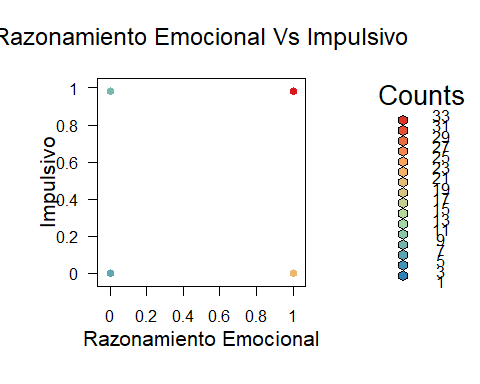
Otro plot similar para ver la relación de ser inhibido e impulsivo

df <- data.frame(matriz.pacientes.datos[, 21], matriz.pacientes.datos[, 24])  
h <- hexbin(df)  
plot(h, colramp=rf, xlab="Inhibido", ylab="Impulsivo", main="Inhibido Vs Impulsivo")



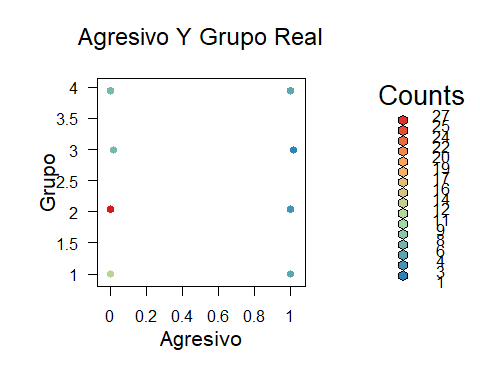
Voy a ver la relación entre el razonamiento emocional (actuar según tus sentimientos) y la impulsividad

df <- data.frame(matriz.pacientes.datos[, 20], matriz.pacientes.datos[, 24])  
h <- hexbin(df)  
plot(h, colramp=rf, xlab="Razonamiento Emocional", ylab="Impulsivo", main="Razonamiento Emocional Vs Impulsivo")



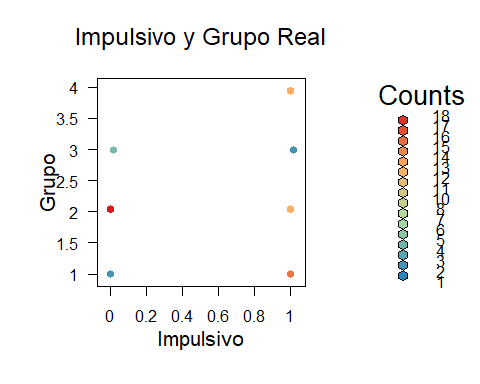
Ahora quiero sacar una relación entre ser agresivo y ver el grupo en el que están

rf <- colorRampPalette(rev(brewer.pal(4,'Spectral')))  
df <- data.frame(matriz.pacientes.datos[, 23], matriz.pacientes.etiquetas[, 25])  
h <- hexbin(df)  
plot(h, colramp=rf, xlab="Agresivo", ylab="Grupo", main="Agresivo Y Grupo Real")



Voy a hacer lo mismo con la impulsividad

rf <- colorRampPalette(rev(brewer.pal(4,'Spectral')))  
df <- data.frame(matriz.pacientes.datos[, 24], matriz.pacientes.etiquetas[, 25])  
h <- hexbin(df)  
plot(h, colramp=rf, xlab="Impulsivo", ylab="Grupo", main="Impulsivo y Grupo Real")



De estas gráficas estamos obteniendo información realmente interesante antes de la predicción de los datos. He preferido hacer gráficas en 2D porque las gráficas en 3D son mucho más difíciles de interpretar que estas bonitas gráficas en 2D

Vamos a ver la correlación que tienen mis variables

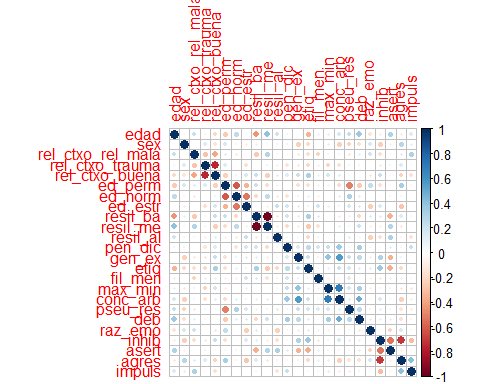
res <- cor(matriz.pacientes.datos[, 1:24], method = "spearman") # Por mi tipo de datos, hacemos la correlación por spearman  
options(width = 100)  
res.round <- round(res, 2)

Como saca una tabla enorme, lo que voy a hacer es usar una librería que me da una función para sacar de una forma bonita las correlaciones entre las variables.

#install.packages("corrplot")  
library(corrplot)

## corrplot 0.84 loaded

corrplot(res.round, method="circle")



Como podemos ver, por ejemplo, resiliencia baja y media tienen una correlación de -1, ya que si hay una no hay la otra y viceversa. Esto pasa igual con las relaciones entre contexto, ya que buena - trauma, trauma - mala, mala - buena tienen que ser inversas.

Como he comentado antes, Lo que voy a hacer ahora es un centrado y escalado de los datos de la matriz. De esta manera, la red neuronal no tendrá ningún valor que destaque especialmente y con ello no dará de inicio más peso a unos valores que a otros, ya que no lo buscamos.

# Modelos de Inteligencia Artificial supervisados

Lo primero que hacemos es importar la librería caret

#install.packages("caret")  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

Ahora hacemos un centrado y escalado de los datos, ya que la edad no sigue el rango del resto de valores, y distorsionaría la predicción

preObjeto <- preProcess(matriz.pacientes.datos, method=c("center", "scale")) # Quiero hacer un centrado y escalado  
matriz.pacientes.datos.centscal <- predict(preObjeto, matriz.pacientes.datos) # Obtengo los valores en la matriz centscal

Ahora vamos a importar la librería nnet, que nos sirve para hacer perceptrones

#install.packages("nnet")  
library(nnet)

Ahora lo que hago es coger un conjunto muy grande de los datos para hacer el entrenamiento

conjuntoEntrenamiento <- sample(1:67, 55)

1 NEURONA

Lo que voy a hacer ahora es entrenar la red neuronal con diferente cantidad de neuronas,y voy a ir comparando el resultado…

SIN SOFTMAX

pacientes.1neu <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=1 )

## # weights: 33  
## initial value 55.729063   
## iter 10 value 39.500282  
## iter 20 value 37.004175  
## iter 30 value 36.722240  
## iter 40 value 36.564012  
## iter 50 value 33.817451  
## iter 60 value 33.795123  
## iter 70 value 33.619692  
## iter 80 value 29.642949  
## iter 90 value 29.353629  
## final value 29.353266   
## converged

Lo voy a entrenar también con el SOFTMAX = true. Esto optimiza la verosimilitud, no el error cuadrático medio…

CON SOFTMAX

pacientes.1neu <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=1, softmax = T )

## # weights: 33  
## initial value 87.584002   
## iter 10 value 59.035626  
## iter 20 value 50.041275  
## iter 30 value 48.457565  
## iter 40 value 48.077963  
## iter 50 value 47.986179  
## iter 60 value 47.979651  
## iter 70 value 47.974518  
## iter 80 value 47.973403  
## final value 47.973252   
## converged

Una vez que lo tengo entrenado, lo que voy a hacer es calcular el error tanto en el entrenamiento como en el test de cada uno

pacientes.prediccion.1neu <- predict( pacientes.1neu, matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.1neu # Vemos las probabilidades de pertenencia de cada valor

## 1 2 3 4  
## 15 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 34 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 30 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 59 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 60 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 3 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 6 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 32 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 23 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 62 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 37 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 17 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 18 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 5 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 26 0.2370370 0.57764482 0.1853182 0.000000e+00  
## 14 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 35 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 65 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 31 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 52 0.7444655 0.15415777 0.1013295 4.717811e-05  
## 1 0.7444655 0.15415791 0.1013296 4.697099e-05  
## 12 0.7444655 0.15415823 0.1013297 4.650559e-05  
## 8 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 55 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 41 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 64 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 48 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 22 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 11 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 24 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 40 0.7444655 0.15415799 0.1013296 4.684559e-05  
## 43 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 57 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 51 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 66 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 54 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 63 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 9 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 46 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 44 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 39 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 13 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 7 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 29 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 45 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 4 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 19 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 16 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 61 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 28 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 33 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 58 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 25 0.3310581 0.06840787 0.0449929 5.555412e-01  
## 50 0.1195680 0.70544725 0.1749848 0.000000e+00  
## 53 0.3310581 0.06840787 0.0449929 5.555412e-01

Ahora que los tengo todos entrenados, Determinamos cual es la máxima, es decir, la clase a la que hay que asignar los objetos

pacientes.prediccion.1neu.class <- apply( pacientes.prediccion.1neu, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.1neu.class

## 15 34 30 59 60 3 6 32 23 62 37 17 18 5 26 14 35 65 31 52 1 12 8 55 41 64 48 22 11 24 40 43 57   
## 2 2 2 4 2 2 4 2 2 2 2 4 2 2 2 4 2 4 2 1 1 1 2 4 4 2 2 2 2 4 1 4 2   
## 51 66 54 63 9 46 44 39 13 7 29 45 4 19 16 61 28 33 58 25 50 53   
## 4 2 2 2 2 2 2 2 2 4 2 2 4 2 4 4 4 2 4 4 2 4

Lo visualizo en forma de tabla para ir viendo el error

table( pacientes.prediccion.1neu.class, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) # Lo vemos en forma de tabla.

##   
## pacientes.prediccion.1neu.class 1 2 3 4  
## 1 4 0 0 0  
## 2 4 23 6 0  
## 4 5 2 1 10

Calculo el acierto

sum( diag( table( pacientes.prediccion.1neu.class, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) ) )/55 # Esta cuenta nos da el índice de acierto

## [1] 0.5090909

TEST

pacientes.prediccion.test.1neu <- predict( pacientes.1neu, matriz.pacientes.datos.centscal[-conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.test.1neu

## 1 2 3 4  
## 2 0.1195680 0.70544725 0.1749848 0.0000000  
## 10 0.3310581 0.06840787 0.0449929 0.5555412  
## 20 0.1195680 0.70544725 0.1749848 0.0000000  
## 21 0.1195680 0.70544725 0.1749848 0.0000000  
## 27 0.3310581 0.06840787 0.0449929 0.5555412  
## 36 0.1195680 0.70544725 0.1749848 0.0000000  
## 38 0.3310581 0.06840787 0.0449929 0.5555412  
## 42 0.3310581 0.06840787 0.0449929 0.5555412  
## 47 0.1195680 0.70544725 0.1749848 0.0000000  
## 49 0.3310581 0.06840787 0.0449929 0.5555412  
## 56 0.3310581 0.06840787 0.0449929 0.5555412  
## 67 0.1195680 0.70544725 0.1749848 0.0000000

pacientes.prediccion.test.1neu.class <- apply( pacientes.prediccion.test.1neu, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.test.1neu.class

## 2 10 20 21 27 36 38 42 47 49 56 67   
## 2 4 2 2 4 2 4 4 2 4 4 2

table( pacientes.prediccion.test.1neu.class , matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] )

##   
## pacientes.prediccion.test.1neu.class 1 2 3 4  
## 2 1 4 0 1  
## 4 3 1 1 1

sum( diag( table( pacientes.prediccion.test.1neu.class, matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] ) ) )/12

## [1] 0.1666667

2 NEURONAS

A partir de ahora voy a hacer exactamente lo mismo, por lo que haré chunks más grandes para evitar una sobrecarga de chunks, y reduciré la cantidad de comentarios, ya que serán redundantes

SIN SOFTMAX

pacientes.2neu <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=2 )

## # weights: 62  
## initial value 73.822795   
## iter 10 value 35.492732  
## iter 20 value 34.286769  
## iter 30 value 34.252324  
## iter 40 value 33.527690  
## iter 50 value 33.364667  
## iter 60 value 33.308029  
## iter 70 value 33.294520  
## iter 80 value 33.288704  
## iter 90 value 33.284713  
## iter 100 value 33.281761  
## final value 33.281761   
## stopped after 100 iterations

CON SOFTMAX

pacientes.2neu <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=2, softmax = T )

## # weights: 62  
## initial value 86.612717   
## iter 10 value 45.472229  
## iter 20 value 35.001757  
## iter 30 value 33.146386  
## iter 40 value 32.927802  
## iter 50 value 32.801936  
## iter 60 value 32.729416  
## iter 70 value 32.667729  
## iter 80 value 32.644965  
## iter 90 value 32.585391  
## iter 100 value 32.565183  
## final value 32.565183   
## stopped after 100 iterations

pacientes.prediccion.2neu <- predict( pacientes.2neu, matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.2neu # Vemos las probabilidades de pertenencia de cada valor

## 1 2 3 4  
## 15 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 34 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 30 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 59 2.214557e-16 5.762004e-18 3.950815e-18 1.000000e+00  
## 60 8.871896e-02 8.429089e-01 6.779056e-02 5.815821e-04  
## 3 8.848800e-02 8.430160e-01 6.807414e-02 4.218520e-04  
## 6 2.214557e-16 5.762004e-18 3.950815e-18 1.000000e+00  
## 32 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 23 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 62 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 37 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 17 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 18 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 5 6.786148e-02 8.339643e-01 9.817427e-02 4.244281e-17  
## 26 2.476203e-01 3.991400e-01 3.532397e-01 2.072925e-43  
## 14 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 35 8.882225e-02 8.428440e-01 6.766095e-02 6.728031e-04  
## 65 2.214557e-16 5.762004e-18 3.950815e-18 1.000000e+00  
## 31 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 52 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 1 9.232483e-01 3.472026e-02 4.107191e-02 9.595509e-04  
## 12 9.232678e-01 3.465410e-02 4.087676e-02 1.201316e-03  
## 8 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 55 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 41 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 64 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 48 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 22 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 11 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 24 2.214557e-16 5.762004e-18 3.950815e-18 1.000000e+00  
## 40 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 43 9.230923e-01 3.484176e-02 4.143972e-02 6.261700e-04  
## 57 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 51 3.179967e-16 8.788452e-18 5.889561e-18 1.000000e+00  
## 66 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 54 2.438659e-01 4.028915e-01 3.532426e-01 1.761141e-43  
## 63 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 9 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 46 7.423202e-02 8.386128e-01 8.715520e-02 8.422245e-13  
## 44 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 39 8.969466e-02 8.412906e-01 6.638445e-02 2.630273e-03  
## 13 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 7 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 29 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 45 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 4 9.392752e-07 9.851755e-07 1.661314e-07 9.999979e-01  
## 19 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 16 2.026167e-15 7.625670e-17 4.545272e-17 1.000000e+00  
## 61 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 28 2.214557e-16 5.762004e-18 3.950815e-18 1.000000e+00  
## 33 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 58 9.232683e-01 3.466209e-02 4.090014e-02 1.169515e-03  
## 25 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 50 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 53 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51

pacientes.prediccion.2neu.class <- apply( pacientes.prediccion.2neu, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.2neu.class

## 15 34 30 59 60 3 6 32 23 62 37 17 18 5 26 14 35 65 31 52 1 12 8 55 41 64 48 22 11 24 40 43 57   
## 2 2 2 4 2 2 4 1 1 1 2 2 2 2 2 1 2 4 2 1 1 1 2 1 2 2 2 2 2 4 1 1 2   
## 51 66 54 63 9 46 44 39 13 7 29 45 4 19 16 61 28 33 58 25 50 53   
## 4 2 2 2 2 2 1 2 2 2 2 2 4 2 4 2 4 2 1 1 2 2

table( pacientes.prediccion.2neu.class, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) # Lo vemos en forma de tabla.

##   
## pacientes.prediccion.2neu.class 1 2 3 4  
## 1 11 1 1 0  
## 2 2 24 6 2  
## 4 0 0 0 8

sum( diag( table( pacientes.prediccion.2neu.class, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) ) )/55 # Esta cuenta nos da el índice de acierto

## [1] 0.6363636

TEST

pacientes.prediccion.test.2neu <- predict( pacientes.2neu, matriz.pacientes.datos.centscal[-conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.test.2neu

## 1 2 3 4  
## 2 3.466469e-02 7.231274e-01 2.422079e-01 6.254032e-51  
## 10 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 20 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 21 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 27 8.029306e-01 5.298269e-02 1.440867e-01 2.966702e-32  
## 36 3.063044e-02 7.357566e-01 2.336129e-01 2.361699e-51  
## 38 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 42 1.866499e-15 4.946072e-17 3.484479e-17 1.000000e+00  
## 47 7.864171e-01 5.446537e-02 1.591176e-01 1.035949e-34  
## 49 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01  
## 56 7.496483e-02 6.764756e-01 5.041182e-02 1.981477e-01  
## 67 7.496528e-02 6.764881e-01 5.041251e-02 1.981341e-01

pacientes.prediccion.test.2neu.class <- apply( pacientes.prediccion.test.2neu, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.test.2neu.class

## 2 10 20 21 27 36 38 42 47 49 56 67   
## 2 2 2 2 1 2 1 4 1 2 2 2

table( pacientes.prediccion.test.2neu.class , matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] )

##   
## pacientes.prediccion.test.2neu.class 1 2 3 4  
## 1 0 0 1 2  
## 2 3 5 0 0  
## 4 1 0 0 0

sum( diag( table( pacientes.prediccion.test.2neu.class, matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] ) ) )/12

## [1] 0.4166667

3 NEURONAS

SIN SOFTMAX

pacientes.3neu <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=3)

## # weights: 91  
## initial value 54.609449   
## iter 10 value 35.479617  
## iter 20 value 34.401186  
## iter 30 value 34.171592  
## iter 40 value 34.131800  
## iter 50 value 33.233890  
## iter 60 value 31.058265  
## iter 70 value 26.066185  
## iter 80 value 25.144329  
## iter 90 value 24.949304  
## iter 100 value 24.898576  
## final value 24.898576   
## stopped after 100 iterations

CON SOFTMAX

pacientes.3neu <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=3, softmax = T)

## # weights: 91  
## initial value 79.918513   
## iter 10 value 46.056204  
## iter 20 value 33.416200  
## iter 30 value 27.059143  
## iter 40 value 24.564542  
## iter 50 value 24.072722  
## iter 60 value 23.984082  
## iter 70 value 23.935546  
## iter 80 value 23.923875  
## iter 90 value 23.918027  
## iter 100 value 23.915293  
## final value 23.915293   
## stopped after 100 iterations

pacientes.prediccion.3neu <- predict( pacientes.3neu, matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.3neu # Vemos las probabilidades de pertenencia de cada valor

## 1 2 3 4  
## 15 0.431902569 0.06808399 5.000134e-01 4.185149e-180  
## 34 0.367429201 0.63248847 8.232724e-05 0.000000e+00  
## 30 0.025287820 0.97471218 0.000000e+00 0.000000e+00  
## 59 0.069515098 0.01881246 2.222120e-01 6.894605e-01  
## 60 0.005216106 0.99478389 0.000000e+00 0.000000e+00  
## 3 0.025287820 0.97471218 0.000000e+00 0.000000e+00  
## 6 0.069515098 0.01881246 2.222120e-01 6.894605e-01  
## 32 0.069515098 0.01881246 2.222120e-01 6.894605e-01  
## 23 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 62 0.037398165 0.58623778 0.000000e+00 3.763641e-01  
## 37 0.130339538 0.86966046 0.000000e+00 0.000000e+00  
## 17 0.037398165 0.58623778 0.000000e+00 3.763641e-01  
## 18 0.037398165 0.58623778 0.000000e+00 3.763641e-01  
## 5 0.005216106 0.99478389 0.000000e+00 0.000000e+00  
## 26 0.000000000 0.00000000 1.000000e+00 0.000000e+00  
## 14 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 35 0.025287820 0.97471218 0.000000e+00 0.000000e+00  
## 65 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 31 0.025287820 0.97471218 0.000000e+00 0.000000e+00  
## 52 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 1 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 12 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 8 0.310275255 0.68972475 8.948736e-144 0.000000e+00  
## 55 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 41 0.069515098 0.01881246 2.222120e-01 6.894605e-01  
## 64 0.005216106 0.99478389 0.000000e+00 0.000000e+00  
## 48 0.005216106 0.99478389 0.000000e+00 0.000000e+00  
## 22 0.431902569 0.06808399 5.000134e-01 4.185149e-180  
## 11 0.037398165 0.58623778 0.000000e+00 3.763641e-01  
## 24 0.069515098 0.01881246 2.222120e-01 6.894605e-01  
## 40 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 43 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 57 0.000000000 0.00000000 1.000000e+00 0.000000e+00  
## 51 0.069515098 0.01881246 2.222120e-01 6.894605e-01  
## 66 0.025287820 0.97471218 0.000000e+00 0.000000e+00  
## 54 0.000000000 0.00000000 1.000000e+00 0.000000e+00  
## 63 0.000000000 0.00000000 1.000000e+00 0.000000e+00  
## 9 0.005216106 0.99478389 0.000000e+00 0.000000e+00  
## 46 0.234491678 0.74290550 0.000000e+00 2.260282e-02  
## 44 0.138912836 0.78821695 0.000000e+00 7.287021e-02  
## 39 0.005216106 0.99478389 0.000000e+00 0.000000e+00  
## 13 0.025287820 0.97471218 0.000000e+00 0.000000e+00  
## 7 0.089353994 0.02438629 7.442358e-06 8.862523e-01  
## 29 0.025287820 0.97471218 0.000000e+00 0.000000e+00  
## 45 0.005216106 0.99478389 0.000000e+00 0.000000e+00  
## 4 0.069515098 0.01881246 2.222120e-01 6.894605e-01  
## 19 0.005216106 0.99478389 0.000000e+00 0.000000e+00  
## 16 0.069515098 0.01881246 2.222120e-01 6.894605e-01  
## 61 0.069515098 0.01881246 2.222120e-01 6.894605e-01  
## 28 0.037398165 0.58623778 0.000000e+00 3.763641e-01  
## 33 0.025287820 0.97471218 0.000000e+00 0.000000e+00  
## 58 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 25 0.869798404 0.04757379 0.000000e+00 8.262780e-02  
## 50 0.005216106 0.99478389 0.000000e+00 0.000000e+00  
## 53 0.234491678 0.74290550 0.000000e+00 2.260282e-02

pacientes.prediccion.3neu.class <- apply( pacientes.prediccion.3neu, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.3neu.class

## 15 34 30 59 60 3 6 32 23 62 37 17 18 5 26 14 35 65 31 52 1 12 8 55 41 64 48 22 11 24 40 43 57   
## 3 2 2 4 2 2 4 4 1 2 2 2 2 2 3 1 2 1 2 1 1 1 2 1 4 2 2 3 2 4 1 1 3   
## 51 66 54 63 9 46 44 39 13 7 29 45 4 19 16 61 28 33 58 25 50 53   
## 4 2 3 3 2 2 2 2 2 4 2 2 4 2 4 4 2 2 1 1 2 2

table( pacientes.prediccion.3neu.class, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) # Lo vemos en forma de tabla.

##   
## pacientes.prediccion.3neu.class 1 2 3 4  
## 1 10 0 0 1  
## 2 2 24 0 2  
## 3 1 0 5 0  
## 4 0 1 2 7

sum( diag( table( pacientes.prediccion.3neu.class, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) ) )/55 # Esta cuenta nos da el índice de acierto

## [1] 0.8363636

TEST

pacientes.prediccion.test.3neu <- predict( pacientes.3neu, matriz.pacientes.datos.centscal[-conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.test.3neu

## 1 2 3 4  
## 2 0.005216106 0.99478389 0.000000 0.0000000  
## 10 0.069515098 0.01881246 0.222212 0.6894605  
## 20 0.025287820 0.97471218 0.000000 0.0000000  
## 21 0.005216106 0.99478389 0.000000 0.0000000  
## 27 0.869798404 0.04757379 0.000000 0.0826278  
## 36 0.000000000 0.00000000 1.000000 0.0000000  
## 38 0.869798404 0.04757379 0.000000 0.0826278  
## 42 0.069515098 0.01881246 0.222212 0.6894605  
## 47 0.069515098 0.01881246 0.222212 0.6894605  
## 49 0.069515098 0.01881246 0.222212 0.6894605  
## 56 0.069515098 0.01881246 0.222212 0.6894605  
## 67 0.037398165 0.58623778 0.000000 0.3763641

pacientes.prediccion.test.3neu.class <- apply( pacientes.prediccion.test.3neu, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.test.3neu.class

## 2 10 20 21 27 36 38 42 47 49 56 67   
## 2 4 2 2 1 3 1 4 4 4 4 2

table( pacientes.prediccion.test.3neu.class , matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] )

##   
## pacientes.prediccion.test.3neu.class 1 2 3 4  
## 1 0 0 1 1  
## 2 1 3 0 0  
## 3 0 1 0 0  
## 4 3 1 0 1

sum( diag( table( pacientes.prediccion.test.3neu.class, matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] ) ) )/12

## [1] 0.3333333

3 NEURONAS

Con Decay

SIN SOFTMAX

pacientes.3neu.decay <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=3, decay = 0.2)

## # weights: 91  
## initial value 68.011332   
## iter 10 value 33.821383  
## iter 20 value 32.546433  
## iter 30 value 32.398699  
## iter 40 value 32.354907  
## iter 50 value 32.340280  
## final value 32.340156   
## converged

CON SOFTMAX

pacientes.3neu.decay <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=3, softmax = T, decay = 0.03)

## # weights: 91  
## initial value 88.401036   
## iter 10 value 44.138159  
## iter 20 value 30.379990  
## iter 30 value 27.753360  
## iter 40 value 27.320567  
## iter 50 value 27.267822  
## iter 60 value 27.265611  
## iter 70 value 27.265085  
## iter 80 value 26.852830  
## iter 90 value 26.305230  
## iter 100 value 25.633698  
## final value 25.633698   
## stopped after 100 iterations

pacientes.prediccion.3neu.decay <- predict( pacientes.3neu.decay, matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.3neu.decay # Vemos las probabilidades de pertenencia de cada valor

## 1 2 3 4  
## 15 0.4441552678 5.054346e-02 0.4648306811 0.0404705893  
## 34 0.2819374235 6.891319e-01 0.0285622588 0.0003684080  
## 30 0.0041149198 8.777990e-01 0.1111276773 0.0069583966  
## 59 0.0305221035 5.933588e-03 0.0373802464 0.9261640621  
## 60 0.0009444723 9.979119e-01 0.0009114961 0.0002321633  
## 3 0.0017549330 9.847844e-01 0.0110087771 0.0024518575  
## 6 0.0327608525 1.755651e-02 0.0257070435 0.9239755989  
## 32 0.0207360457 3.926160e-01 0.2026330291 0.3840148821  
## 23 0.9943865168 2.359444e-03 0.0023670794 0.0008869601  
## 62 0.2380849783 4.866553e-01 0.0022425014 0.2730171794  
## 37 0.3000322180 6.670165e-01 0.0325554027 0.0003959115  
## 17 0.0519509721 1.215172e-01 0.0491394976 0.7773923074  
## 18 0.0090167062 9.854310e-01 0.0008015139 0.0047507501  
## 5 0.0007861514 9.986601e-01 0.0003973706 0.0001563299  
## 26 0.0792098025 1.105753e-02 0.9084127938 0.0013198747  
## 14 0.8319104943 5.206143e-05 0.1153799330 0.0526575112  
## 35 0.0053754486 9.792750e-01 0.0137000108 0.0016495123  
## 65 0.0497239517 1.004695e-02 0.0770481899 0.8631809069  
## 31 0.0267592615 9.567173e-01 0.0012366419 0.0152868444  
## 52 0.8005885957 1.616341e-01 0.0058815168 0.0318957591  
## 1 0.8402441710 7.549236e-02 0.0547797329 0.0294837365  
## 12 0.2764089077 4.284980e-01 0.0018019913 0.2932911088  
## 8 0.2856122229 6.848639e-01 0.0291519460 0.0003719099  
## 55 0.7815690592 1.064287e-04 0.1764114726 0.0419130395  
## 41 0.1511627678 1.292721e-01 0.6394449784 0.0801201604  
## 64 0.0015386413 9.859115e-01 0.0088984852 0.0036513344  
## 48 0.0083787923 9.848155e-01 0.0011943535 0.0056113836  
## 22 0.4441552678 5.054346e-02 0.4648306811 0.0404705893  
## 11 0.2715667646 4.108951e-01 0.0019990780 0.3155390882  
## 24 0.3918952975 3.348356e-01 0.0022453491 0.2710237282  
## 40 0.9588263606 1.287444e-02 0.0017032632 0.0265959374  
## 43 0.9960429628 5.945260e-04 0.0007767248 0.0025857864  
## 57 0.1619754072 2.643990e-02 0.8101902641 0.0013944262  
## 51 0.4629424010 3.099557e-01 0.0025191861 0.2245827627  
## 66 0.0014092716 9.948949e-01 0.0030940461 0.0006017765  
## 54 0.0745305275 8.277684e-03 0.9159552550 0.0012365333  
## 63 0.0577591776 3.453362e-02 0.9050922343 0.0026149672  
## 9 0.0012107283 8.060629e-01 0.1749350865 0.0177912636  
## 46 0.0540064483 8.890909e-01 0.0365308103 0.0203718659  
## 44 0.2854985249 4.104674e-01 0.0019445664 0.3020894793  
## 39 0.0007089239 9.987796e-01 0.0003707309 0.0001407416  
## 13 0.0179530802 9.783862e-01 0.0031761319 0.0004845836  
## 7 0.0245652523 3.182338e-02 0.0815938114 0.8620175536  
## 29 0.0028671055 9.881820e-01 0.0080583990 0.0008924638  
## 45 0.0011793316 8.135581e-01 0.1698682764 0.0153942432  
## 4 0.0251558515 1.023907e-02 0.0246246744 0.9399803999  
## 19 0.0608253869 8.697478e-01 0.0492370219 0.0201898288  
## 16 0.0221276202 7.304391e-03 0.0322502725 0.9383177161  
## 61 0.2830989981 3.994647e-01 0.0020778179 0.3153584976  
## 28 0.0369624439 1.951585e-02 0.0224067878 0.9211149183  
## 33 0.0006120587 9.986404e-01 0.0005758876 0.0001716790  
## 58 0.9629111640 1.601601e-03 0.0308993348 0.0045878997  
## 25 0.9583894452 1.222267e-02 0.0018407346 0.0275471523  
## 50 0.0583805487 8.472634e-01 0.0856043822 0.0087516414  
## 53 0.0125815693 8.954214e-01 0.0810092163 0.0109878001

pacientes.prediccion.3neu.class.decay <- apply( pacientes.prediccion.3neu.decay, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.3neu.class.decay

## 15 34 30 59 60 3 6 32 23 62 37 17 18 5 26 14 35 65 31 52 1 12 8 55 41 64 48 22 11 24 40 43 57   
## 3 2 2 4 2 2 4 2 1 2 2 4 2 2 3 1 2 4 2 1 1 2 2 1 3 2 2 3 2 1 1 1 3   
## 51 66 54 63 9 46 44 39 13 7 29 45 4 19 16 61 28 33 58 25 50 53   
## 1 2 3 3 2 2 2 2 2 4 2 2 4 2 4 2 4 2 1 1 2 2

table( pacientes.prediccion.3neu.class.decay, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) # Lo vemos en forma de tabla.

##   
## pacientes.prediccion.3neu.class.decay 1 2 3 4  
## 1 9 0 0 2  
## 2 3 25 1 0  
## 3 1 0 6 0  
## 4 0 0 0 8

sum( diag( table( pacientes.prediccion.3neu.class.decay, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) ) )/55 # Esta cuenta nos da el índice de acierto

## [1] 0.8727273

TEST

pacientes.prediccion.test.3neu.decay <- predict( pacientes.3neu.decay, matriz.pacientes.datos.centscal[-conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.test.3neu.decay

## 1 2 3 4  
## 2 0.983765423 4.029230e-03 0.001264362 0.0109409845  
## 10 0.022386765 2.771315e-02 0.069412915 0.8804871674  
## 20 0.388229349 5.784127e-01 0.033002679 0.0003552665  
## 21 0.001091909 8.166265e-01 0.166754092 0.0155275256  
## 27 0.982206553 4.063501e-04 0.011931096 0.0054560003  
## 36 0.642634194 2.536031e-01 0.103294207 0.0004685358  
## 38 0.909282764 7.284935e-05 0.056611741 0.0340326456  
## 42 0.996212700 1.072424e-03 0.001340614 0.0013742620  
## 47 0.014826836 7.694503e-01 0.090409639 0.1253132543  
## 49 0.285083895 4.124383e-01 0.001932884 0.3005449014  
## 56 0.864811955 3.297144e-04 0.088424414 0.0464339167  
## 67 0.467493542 2.862088e-01 0.003398532 0.2428990877

pacientes.prediccion.test.3neu.class.decay <- apply( pacientes.prediccion.test.3neu.decay, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.test.3neu.class.decay

## 2 10 20 21 27 36 38 42 47 49 56 67   
## 1 4 2 2 1 1 1 1 2 2 1 1

table( pacientes.prediccion.test.3neu.class.decay , matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] )

##   
## pacientes.prediccion.test.3neu.class.decay 1 2 3 4  
## 1 3 2 1 1  
## 2 1 2 0 1  
## 4 0 1 0 0

sum( diag( table( pacientes.prediccion.test.3neu.class.decay, matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] ) ) )/12

## [1] 0.4166667

5 NEURONAS

SIN SOFTMAX

pacientes.5neu <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=5 )

## # weights: 149  
## initial value 59.114081   
## iter 10 value 54.984264  
## iter 20 value 54.970766  
## iter 30 value 54.932572  
## iter 40 value 54.440845  
## iter 50 value 34.433756  
## iter 60 value 31.302974  
## iter 70 value 31.247876  
## iter 80 value 31.155865  
## iter 90 value 29.719713  
## iter 100 value 28.943488  
## final value 28.943488   
## stopped after 100 iterations

CON SOFTMAX

pacientes.5neu <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=5, softmax = T )

## # weights: 149  
## initial value 114.140007   
## iter 10 value 29.777231  
## iter 20 value 15.457227  
## iter 30 value 11.031680  
## iter 40 value 9.200256  
## iter 50 value 8.554918  
## iter 60 value 7.785991  
## iter 70 value 6.564343  
## iter 80 value 6.484720  
## iter 90 value 6.411209  
## iter 100 value 6.408272  
## final value 6.408272   
## stopped after 100 iterations

pacientes.prediccion.5neu <- predict( pacientes.5neu, matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.5neu # Vemos las probabilidades de pertenencia de cada valor

## 1 2 3 4  
## 15 5.002334e-01 0.000000e+00 4.997666e-01 0.000000e+00  
## 34 2.498410e-01 7.501590e-01 0.000000e+00 0.000000e+00  
## 30 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 59 0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00  
## 60 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 3 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 6 0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00  
## 32 2.528406e-26 2.350098e-05 9.999765e-01 0.000000e+00  
## 23 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  
## 62 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  
## 37 2.498410e-01 7.501590e-01 0.000000e+00 0.000000e+00  
## 17 0.000000e+00 5.000167e-01 0.000000e+00 4.999833e-01  
## 18 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 5 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 26 0.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00  
## 14 1.000000e+00 0.000000e+00 4.741402e-224 0.000000e+00  
## 35 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 65 2.130268e-06 0.000000e+00 2.174668e-266 9.999979e-01  
## 31 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 52 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  
## 1 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  
## 12 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  
## 8 2.498410e-01 7.501590e-01 0.000000e+00 0.000000e+00  
## 55 1.000000e+00 0.000000e+00 4.741402e-224 0.000000e+00  
## 41 1.533678e-167 1.550578e-85 1.000000e+00 0.000000e+00  
## 64 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 48 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 22 5.002334e-01 0.000000e+00 4.997666e-01 0.000000e+00  
## 11 0.000000e+00 5.000167e-01 0.000000e+00 4.999833e-01  
## 24 0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00  
## 40 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  
## 43 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  
## 57 1.533678e-167 1.550578e-85 1.000000e+00 0.000000e+00  
## 51 0.000000e+00 1.630783e-09 0.000000e+00 1.000000e+00  
## 66 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 54 0.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00  
## 63 0.000000e+00 4.994559e-01 5.005441e-01 0.000000e+00  
## 9 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 46 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 44 0.000000e+00 9.999972e-01 2.782333e-06 0.000000e+00  
## 39 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 13 2.498410e-01 7.501590e-01 0.000000e+00 0.000000e+00  
## 7 0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00  
## 29 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 45 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 4 0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00  
## 19 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 16 0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00  
## 61 0.000000e+00 9.999814e-01 1.862115e-05 3.012053e-17  
## 28 0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00  
## 33 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 58 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  
## 25 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  
## 50 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00  
## 53 0.000000e+00 4.994559e-01 5.005441e-01 0.000000e+00

pacientes.prediccion.5neu.class <- apply( pacientes.prediccion.5neu, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.5neu.class

## 15 34 30 59 60 3 6 32 23 62 37 17 18 5 26 14 35 65 31 52 1 12 8 55 41 64 48 22 11 24 40 43 57   
## 1 2 2 4 2 2 4 3 1 1 2 2 2 2 3 1 2 4 2 1 1 1 2 1 3 2 2 1 2 4 1 1 3   
## 51 66 54 63 9 46 44 39 13 7 29 45 4 19 16 61 28 33 58 25 50 53   
## 4 2 3 3 2 2 2 2 2 4 2 2 4 2 4 2 4 2 1 1 2 3

table( pacientes.prediccion.5neu.class, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) # Lo vemos en forma de tabla.

##   
## pacientes.prediccion.5neu.class 1 2 3 4  
## 1 12 0 1 0  
## 2 1 24 0 1  
## 3 0 1 6 0  
## 4 0 0 0 9

sum( diag( table( pacientes.prediccion.5neu.class, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) ) )/55 # Esta cuenta nos da el índice de acierto

## [1] 0.9272727

TEST

pacientes.prediccion.test.5neu <- predict( pacientes.5neu, matriz.pacientes.datos.centscal[-conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.test.5neu

## 1 2 3 4  
## 2 0.000000e+00 9.999972e-01 2.782333e-06 0  
## 10 5.002334e-01 0.000000e+00 4.997666e-01 0  
## 20 0.000000e+00 1.000000e+00 0.000000e+00 0  
## 21 0.000000e+00 1.000000e+00 0.000000e+00 0  
## 27 1.533678e-167 1.550578e-85 1.000000e+00 0  
## 36 2.498410e-01 7.501590e-01 0.000000e+00 0  
## 38 1.000000e+00 0.000000e+00 4.741402e-224 0  
## 42 1.000000e+00 0.000000e+00 0.000000e+00 0  
## 47 1.000000e+00 0.000000e+00 0.000000e+00 0  
## 49 1.533678e-167 1.550578e-85 1.000000e+00 0  
## 56 0.000000e+00 0.000000e+00 1.000000e+00 0  
## 67 0.000000e+00 1.000000e+00 0.000000e+00 0

pacientes.prediccion.test.5neu.class <- apply( pacientes.prediccion.test.5neu, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.test.5neu.class

## 2 10 20 21 27 36 38 42 47 49 56 67   
## 2 1 2 2 3 2 1 1 1 3 3 2

table( pacientes.prediccion.test.5neu.class , matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] )

##   
## pacientes.prediccion.test.5neu.class 1 2 3 4  
## 1 1 1 1 1  
## 2 1 4 0 0  
## 3 2 0 0 1

sum( diag( table( pacientes.prediccion.test.5neu.class, matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] ) ) )/12

## [1] 0.4166667

5 NEURONAS

CON DECAY

SIN SOFTMAX

pacientes.5neu.decay <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=5, decay=0.1)

## # weights: 149  
## initial value 67.440251   
## iter 10 value 30.592841  
## iter 20 value 26.284499  
## iter 30 value 25.604709  
## iter 40 value 25.146994  
## iter 50 value 25.072643  
## iter 60 value 24.806778  
## iter 70 value 24.776431  
## iter 80 value 24.774841  
## iter 90 value 24.774411  
## final value 24.774379   
## converged

CON SOFTMAX

pacientes.5neu.decay <- nnet( matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], class.ind( matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) , size=5, softmax = T, decay = 0.05)

## # weights: 149  
## initial value 80.598165   
## iter 10 value 38.098295  
## iter 20 value 27.470244  
## iter 30 value 24.919379  
## iter 40 value 23.717215  
## iter 50 value 23.428744  
## iter 60 value 23.422650  
## iter 70 value 23.420847  
## iter 80 value 23.419565  
## final value 23.419562   
## converged

pacientes.prediccion.5neu.decay <- predict( pacientes.5neu.decay, matriz.pacientes.datos.centscal[conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.5neu.decay # Vemos las probabilidades de pertenencia de cada valor

## 1 2 3 4  
## 15 0.4641281134 4.755782e-02 0.4584580114 2.985605e-02  
## 34 0.4231451929 5.494510e-01 0.0180836575 9.320186e-03  
## 30 0.0399925433 9.241283e-01 0.0344500596 1.429079e-03  
## 59 0.1310527729 1.371181e-03 0.0063847245 8.611913e-01  
## 60 0.0011464862 9.616220e-01 0.0179194579 1.931201e-02  
## 3 0.0612952445 9.137249e-01 0.0223695532 2.610352e-03  
## 6 0.0353325205 5.048642e-02 0.0532979023 8.608832e-01  
## 32 0.0134255741 3.290426e-01 0.4460419912 2.114898e-01  
## 23 0.9921938483 7.121778e-03 0.0004144118 2.699622e-04  
## 62 0.4421809668 5.074785e-01 0.0050544325 4.528608e-02  
## 37 0.1452673630 8.153208e-01 0.0349484131 4.463383e-03  
## 17 0.0454373530 2.118317e-01 0.0027458889 7.399850e-01  
## 18 0.0083831880 8.131856e-01 0.0585213940 1.199099e-01  
## 5 0.0169732731 9.657113e-01 0.0168716848 4.437840e-04  
## 26 0.0327575953 3.819105e-02 0.9202288964 8.822454e-03  
## 14 0.8744442801 9.706274e-05 0.0373245317 8.813413e-02  
## 35 0.0398388141 8.046603e-01 0.1422380946 1.326277e-02  
## 65 0.0905711656 8.930670e-02 0.0100328856 8.100893e-01  
## 31 0.0130582309 9.848063e-01 0.0003021563 1.833322e-03  
## 52 0.7457276079 2.166612e-01 0.0098542349 2.775699e-02  
## 1 0.6443096570 2.810260e-02 0.1162715655 2.113162e-01  
## 12 0.8885390786 2.644859e-02 0.0007558347 8.425650e-02  
## 8 0.3465473141 6.237951e-01 0.0214593314 8.198295e-03  
## 55 0.9837059322 3.719018e-03 0.0033749926 9.200057e-03  
## 41 0.2218344821 9.298216e-02 0.6425604138 4.262294e-02  
## 64 0.0028361013 9.817918e-01 0.0019163582 1.345571e-02  
## 48 0.0341025822 9.296611e-01 0.0298805373 6.355795e-03  
## 22 0.4641281134 4.755782e-02 0.4584580114 2.985605e-02  
## 11 0.0051990825 9.013281e-01 0.0050227308 8.845008e-02  
## 24 0.1468217325 9.750284e-03 0.0053329797 8.380950e-01  
## 40 0.9438531407 2.078975e-02 0.0002732685 3.508384e-02  
## 43 0.8794004453 3.670436e-02 0.0023130374 8.158216e-02  
## 57 0.0041544918 1.267828e-01 0.8681257713 9.369789e-04  
## 51 0.0950432802 1.379851e-01 0.1264672685 6.405044e-01  
## 66 0.0114054176 9.795423e-01 0.0062826938 2.769625e-03  
## 54 0.0256089815 2.837171e-03 0.9692185875 2.335260e-03  
## 63 0.0246648332 1.575661e-01 0.8144947874 3.274255e-03  
## 9 0.0003343515 9.536906e-01 0.0298581844 1.611690e-02  
## 46 0.0207286093 9.019289e-01 0.0189945970 5.834792e-02  
## 44 0.5008701097 4.243250e-01 0.0059805854 6.882434e-02  
## 39 0.0039856122 9.921853e-01 0.0036845242 1.445577e-04  
## 13 0.0485766549 9.264023e-01 0.0233969900 1.624008e-03  
## 7 0.0320915658 4.940572e-02 0.0205589741 8.979437e-01  
## 29 0.0119811515 9.816830e-01 0.0058104325 5.254273e-04  
## 45 0.0027452648 9.930388e-01 0.0039106747 3.052759e-04  
## 4 0.0357082230 1.952785e-02 0.0028793129 9.418846e-01  
## 19 0.0035245322 8.319066e-01 0.0944130634 7.015578e-02  
## 16 0.0238839817 9.152642e-03 0.0122803407 9.546830e-01  
## 61 0.0116963209 6.998557e-01 0.0503814646 2.380665e-01  
## 28 0.0336073091 8.866371e-02 0.0408888229 8.368402e-01  
## 33 0.0034170845 9.934782e-01 0.0030661182 3.860245e-05  
## 58 0.8510560721 2.185408e-04 0.0487030737 1.000223e-01  
## 25 0.9403292441 4.676570e-02 0.0046081306 8.296923e-03  
## 50 0.0654721487 8.593960e-01 0.0420313459 3.310055e-02  
## 53 0.0401832498 9.422409e-01 0.0093157441 8.260119e-03

pacientes.prediccion.5neu.decay.class <- apply( pacientes.prediccion.5neu.decay, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.5neu.decay.class

## 15 34 30 59 60 3 6 32 23 62 37 17 18 5 26 14 35 65 31 52 1 12 8 55 41 64 48 22 11 24 40 43 57   
## 1 2 2 4 2 2 4 3 1 2 2 4 2 2 3 1 2 4 2 1 1 1 2 1 3 2 2 1 2 4 1 1 3   
## 51 66 54 63 9 46 44 39 13 7 29 45 4 19 16 61 28 33 58 25 50 53   
## 4 2 3 3 2 2 1 2 2 4 2 2 4 2 4 2 4 2 1 1 2 2

table( pacientes.prediccion.5neu.decay.class, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) # Lo vemos en forma de tabla.

##   
## pacientes.prediccion.5neu.decay.class 1 2 3 4  
## 1 11 1 1 0  
## 2 2 24 0 0  
## 3 0 0 6 0  
## 4 0 0 0 10

sum( diag( table( pacientes.prediccion.5neu.decay.class, matriz.pacientes.etiquetas[conjuntoEntrenamiento, 25] ) ) )/55 # Esta cuenta nos da el índice de acierto

## [1] 0.9272727

TEST

pacientes.prediccion.test.decay.5neu <- predict( pacientes.5neu.decay, matriz.pacientes.datos.centscal[-conjuntoEntrenamiento, 1:24], type="raw" )  
pacientes.prediccion.test.decay.5neu

## 1 2 3 4  
## 2 0.8822583626 0.098170686 0.004611395 0.0149595567  
## 10 0.0159615073 0.283219316 0.089247458 0.6115717188  
## 20 0.0005640989 0.987877662 0.011307869 0.0002503699  
## 21 0.0008515496 0.893108124 0.017862246 0.0881780801  
## 27 0.7063077743 0.008795777 0.176197798 0.1086986510  
## 36 0.9289200126 0.049784338 0.016386956 0.0049086932  
## 38 0.5250412928 0.017047369 0.027216733 0.4306946055  
## 42 0.0909143595 0.036390742 0.144381781 0.7283131177  
## 47 0.0512954274 0.363141742 0.319575536 0.2659872941  
## 49 0.0078391580 0.783515404 0.043028933 0.1656165050  
## 56 0.0472786739 0.181023526 0.112789489 0.6589083110  
## 67 0.1834007124 0.103901976 0.022009192 0.6906881200

pacientes.prediccion.test.decay.5neu.class <- apply( pacientes.prediccion.test.decay.5neu, MARGIN=1, FUN='which.is.max')  
pacientes.prediccion.test.decay.5neu.class

## 2 10 20 21 27 36 38 42 47 49 56 67   
## 1 4 2 2 1 1 1 4 2 2 4 4

table( pacientes.prediccion.test.decay.5neu.class , matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] )

##   
## pacientes.prediccion.test.decay.5neu.class 1 2 3 4  
## 1 0 2 1 1  
## 2 1 2 0 1  
## 4 3 1 0 0

sum( diag( table( pacientes.prediccion.test.decay.5neu.class, matriz.pacientes.etiquetas[-conjuntoEntrenamiento, 25] ) ) )/12

## [1] 0.1666667

# Obtención de Resultados de Perceptrón

Importo los datos:

dataset.resultados <- read.csv2("C:/Users/jorge/Desktop/Documentos Clase/Universidad/4ºCarrera/1er Cuatrimestre/Inteligencia Artificial/Trabajo Fin de Asignatura/Resultados.txt")

Ahora voy a sacar un gráfico donde comparo los resultados.

#install.packages("plotly")  
library("plotly")

##   
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':  
##   
## last\_plot

## The following object is masked from 'package:stats':  
##   
## filter

## The following object is masked from 'package:graphics':  
##   
## layout

tipos = dataset.resultados[, 1]  
real = dataset.resultados[, 2]  
practico = dataset.resultados[, 3]  
  
p<- plot\_ly(dataset.resultados, x = ~tipos, y = ~real, type = 'bar', name = 'Real') %>% add\_trace(y = ~practico, name = 'Práctico') %>% layout(yaxis = list(title = 'Porcentaje'), barmode = 'group')  
  
p

## PhantomJS not found. You can install it with webshot::install\_phantomjs(). If it is installed, please make sure the phantomjs executable can be found via the PATH variable.

#Mostramos el gráfico interactivo

## KNN

#install.packages("class")  
library("class")  
  
# Para hacer la predicción con knn, voy a coger los grupos de una manera distinta:  
  
conjuntoEntrenamiento = matriz.pacientes.datos.centscal[1:55, 1:24]  
conjuntoTest = matriz.pacientes.datos.centscal[56:67, 1:24]  
# Utilizo por supuesto la matriz de centrado y escalado  
  
etiquetasEntrenamiento = matriz.pacientes.etiquetas[1:55, 25]  
etiquetasTest = matriz.pacientes.etiquetas[56:67, 25]

conjuntoEntrenamiento  
conjuntoTest  
etiquetasEntrenamiento  
etiquetasTest

Para K = 8…

# K = 8  
  
prediccion.knn.8 <- knn(train = conjuntoEntrenamiento, test = conjuntoTest, cl = etiquetasEntrenamiento, k = 8)  
prediccion.knn.8

## [1] 1 2 2 1 2 1 2 2 4 1 2 2  
## Levels: 1 2 3 4

Sacamos crosstable:

#install.packages("gmodels")  
library("gmodels")

CrossTable(x = etiquetasTest , y = prediccion.knn.8, prop.chisq = FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 12   
##   
##   
## | prediccion.knn.8   
## etiquetasTest | 1 | 2 | 4 | Row Total |   
## --------------|-----------|-----------|-----------|-----------|  
## 1 | 1 | 3 | 0 | 4 |   
## | 0.250 | 0.750 | 0.000 | 0.333 |   
## | 0.250 | 0.429 | 0.000 | |   
## | 0.083 | 0.250 | 0.000 | |   
## --------------|-----------|-----------|-----------|-----------|  
## 2 | 1 | 2 | 1 | 4 |   
## | 0.250 | 0.500 | 0.250 | 0.333 |   
## | 0.250 | 0.286 | 1.000 | |   
## | 0.083 | 0.167 | 0.083 | |   
## --------------|-----------|-----------|-----------|-----------|  
## 3 | 0 | 2 | 0 | 2 |   
## | 0.000 | 1.000 | 0.000 | 0.167 |   
## | 0.000 | 0.286 | 0.000 | |   
## | 0.000 | 0.167 | 0.000 | |   
## --------------|-----------|-----------|-----------|-----------|  
## 4 | 2 | 0 | 0 | 2 |   
## | 1.000 | 0.000 | 0.000 | 0.167 |   
## | 0.500 | 0.000 | 0.000 | |   
## | 0.167 | 0.000 | 0.000 | |   
## --------------|-----------|-----------|-----------|-----------|  
## Column Total | 4 | 7 | 1 | 12 |   
## | 0.333 | 0.583 | 0.083 | |   
## --------------|-----------|-----------|-----------|-----------|  
##   
##

Para K = 6

# K = 6  
  
prediccion.knn.6 <- knn(train = conjuntoEntrenamiento, test = conjuntoTest, cl = etiquetasEntrenamiento, k = 6)  
prediccion.knn.6

## [1] 1 2 2 1 2 1 2 2 2 1 2 2  
## Levels: 1 2 3 4

CrossTable(x = etiquetasTest , y = prediccion.knn.6, prop.chisq = FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 12   
##   
##   
## | prediccion.knn.6   
## etiquetasTest | 1 | 2 | Row Total |   
## --------------|-----------|-----------|-----------|  
## 1 | 1 | 3 | 4 |   
## | 0.250 | 0.750 | 0.333 |   
## | 0.250 | 0.375 | |   
## | 0.083 | 0.250 | |   
## --------------|-----------|-----------|-----------|  
## 2 | 1 | 3 | 4 |   
## | 0.250 | 0.750 | 0.333 |   
## | 0.250 | 0.375 | |   
## | 0.083 | 0.250 | |   
## --------------|-----------|-----------|-----------|  
## 3 | 0 | 2 | 2 |   
## | 0.000 | 1.000 | 0.167 |   
## | 0.000 | 0.250 | |   
## | 0.000 | 0.167 | |   
## --------------|-----------|-----------|-----------|  
## 4 | 2 | 0 | 2 |   
## | 1.000 | 0.000 | 0.167 |   
## | 0.500 | 0.000 | |   
## | 0.167 | 0.000 | |   
## --------------|-----------|-----------|-----------|  
## Column Total | 4 | 8 | 12 |   
## | 0.333 | 0.667 | |   
## --------------|-----------|-----------|-----------|  
##   
##

Para k = 10

# K = 10  
  
prediccion.knn.10 <- knn(train = conjuntoEntrenamiento, test = conjuntoTest, cl = etiquetasEntrenamiento, k = 10)  
prediccion.knn.10

## [1] 1 2 1 1 2 1 2 2 4 1 2 2  
## Levels: 1 2 3 4

CrossTable(x = etiquetasTest , y = prediccion.knn.10, prop.chisq = FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 12   
##   
##   
## | prediccion.knn.10   
## etiquetasTest | 1 | 2 | 4 | Row Total |   
## --------------|-----------|-----------|-----------|-----------|  
## 1 | 2 | 2 | 0 | 4 |   
## | 0.500 | 0.500 | 0.000 | 0.333 |   
## | 0.400 | 0.333 | 0.000 | |   
## | 0.167 | 0.167 | 0.000 | |   
## --------------|-----------|-----------|-----------|-----------|  
## 2 | 1 | 2 | 1 | 4 |   
## | 0.250 | 0.500 | 0.250 | 0.333 |   
## | 0.200 | 0.333 | 1.000 | |   
## | 0.083 | 0.167 | 0.083 | |   
## --------------|-----------|-----------|-----------|-----------|  
## 3 | 0 | 2 | 0 | 2 |   
## | 0.000 | 1.000 | 0.000 | 0.167 |   
## | 0.000 | 0.333 | 0.000 | |   
## | 0.000 | 0.167 | 0.000 | |   
## --------------|-----------|-----------|-----------|-----------|  
## 4 | 2 | 0 | 0 | 2 |   
## | 1.000 | 0.000 | 0.000 | 0.167 |   
## | 0.400 | 0.000 | 0.000 | |   
## | 0.167 | 0.000 | 0.000 | |   
## --------------|-----------|-----------|-----------|-----------|  
## Column Total | 5 | 6 | 1 | 12 |   
## | 0.417 | 0.500 | 0.083 | |   
## --------------|-----------|-----------|-----------|-----------|  
##   
##

### Como se puede observar, la mejor predicción la hemos hecho con K = 8

## Random Forest

Ahora voy a implementar una solución mediante Random Forest:

Lo primero que hacemos es importar el paquete de Random Forest

#install.packages("randomForest")  
library(randomForest)

## Warning: package 'randomForest' was built under R version 3.5.2

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

Una vez instalado e importado, lo que tengo que hacer es crear el Random Forest, y ejecutarlo…

model <- randomForest(grupo ~ ., data = dataset[2:26], importance = TRUE)

## Warning in randomForest.default(m, y, ...): The response has five or fewer unique values. Are you  
## sure you want to do regression?

model

##   
## Call:  
## randomForest(formula = grupo ~ ., data = dataset[2:26], importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 8  
##   
## Mean of squared residuals: 1.283642  
## % Var explained: -23.49

Ahora lo voy a hacer con 10 fold X Validation:

result <- rfcv(dataset[2:26], dataset$grupo, cv.fold=10)

## Warning in randomForest.default(trainx[idx != i, , drop = FALSE], trainy[idx != : The response has  
## five or fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, , drop = FALSE], trainy[idx != : The response has  
## five or fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, , drop = FALSE], trainy[idx != : The response has  
## five or fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, , drop = FALSE], trainy[idx != : The response has  
## five or fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, , drop = FALSE], trainy[idx != : The response has  
## five or fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, , drop = FALSE], trainy[idx != : The response has  
## five or fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, , drop = FALSE], trainy[idx != : The response has  
## five or fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, , drop = FALSE], trainy[idx != : The response has  
## five or fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, , drop = FALSE], trainy[idx != : The response has  
## five or fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, , drop = FALSE], trainy[idx != : The response has  
## five or fewer unique values. Are you sure you want to do regression?

## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?  
  
## Warning in randomForest.default(trainx[idx != i, imp.idx, drop = FALSE], : The response has five or  
## fewer unique values. Are you sure you want to do regression?

result

## $n.var  
## [1] 25 12 6 3 1  
##   
## $error.cv  
## 25 12 6 3 1   
## 3.391799e-01 2.552296e-01 1.909404e-01 2.689975e-01 5.373134e-07   
##   
## $predicted  
## $predicted$`25`  
## [1] 1.932425 1.873850 1.934950 3.337756 1.721599 3.210449 2.927183 1.615045 2.109428 2.169623  
## [11] 2.011059 1.910429 1.850162 2.309648 2.021316 3.130194 2.826287 1.822460 2.067767 2.059370  
## [21] 2.179930 2.737553 1.582802 2.738642 1.730212 2.717493 3.011703 3.070571 2.091698 1.900269  
## [31] 1.872288 3.181815 1.839978 1.846715 2.027419 1.882813 1.974365 2.950831 1.744233 1.575861  
## [41] 2.853614 1.649973 1.549764 1.894096 1.929231 1.992615 3.272996 1.914428 1.815930 2.015915  
## [51] 2.808407 1.507972 1.994642 2.946243 1.838062 1.655721 2.759243 1.735528 3.113943 2.041806  
## [61] 1.954111 1.788166 2.708756 2.285385 3.102144 1.918739 1.748350  
##   
## $predicted$`12`  
## [1] 1.763428 1.911038 1.987537 3.528695 1.881770 3.402074 3.125803 1.573659 2.102284 2.167280  
## [11] 1.998871 1.759911 1.845226 2.410259 1.902383 3.313740 2.832927 1.811008 2.105495 2.044333  
## [21] 2.213584 2.761336 1.624606 2.867555 1.706419 2.890360 3.142116 2.992395 2.091413 1.964243  
## [31] 1.800470 3.207373 1.891558 1.880332 2.037092 1.969847 2.000641 2.966553 1.931202 1.500796  
## [41] 2.946804 1.328386 1.441651 1.822842 1.878582 2.015479 3.442056 2.007124 1.474193 2.024052  
## [51] 3.074621 1.297311 2.137498 2.890895 1.541554 1.579780 2.804805 1.683836 3.365304 2.046069  
## [61] 1.805105 1.697897 2.859862 2.315216 3.231952 1.872054 1.654232  
##   
## $predicted$`6`  
## [1] 1.503098 1.791666 2.014426 3.653294 1.689380 3.432207 3.225261 1.518407 2.055759 2.155731  
## [11] 1.870160 1.485658 1.936981 2.267856 1.796398 3.379255 2.937657 1.890367 1.962209 2.015429  
## [21] 2.279459 2.977979 1.543998 2.902308 1.508784 2.855469 3.029567 3.296403 2.146767 1.912381  
## [31] 1.873645 3.160881 1.838503 1.929713 1.980213 2.006114 2.044204 2.973991 1.689380 1.422031  
## [41] 3.103664 1.290992 1.422031 1.854099 1.824786 1.881621 3.585664 1.960992 1.315730 1.981018  
## [51] 3.247212 1.262781 1.787574 2.938540 1.601482 1.360051 2.827114 1.591127 3.484821 1.981018  
## [61] 1.861676 1.477421 2.884354 2.016291 3.247130 1.968068 1.310979  
##   
## $predicted$`3`  
## [1] 1.648238 1.731279 2.059809 3.335562 1.731279 3.530832 3.073106 1.818345 2.045861 2.022962  
## [11] 1.936114 1.775693 1.936800 1.869642 1.659477 3.438509 2.981877 1.883066 1.916245 2.049112  
## [21] 2.276626 2.899144 1.818345 2.711279 1.600467 2.831087 3.284968 3.055362 2.089403 2.017047  
## [31] 1.930834 2.893831 2.070696 2.025985 2.045861 2.049112 2.072068 2.741560 1.731279 1.600467  
## [41] 2.976609 1.630562 1.600467 2.045861 2.070696 1.932034 3.210791 1.906495 1.494501 2.035150  
## [51] 2.917954 1.580796 1.822866 2.671584 1.782707 1.664716 2.697113 1.640166 3.210791 2.035150  
## [61] 2.002562 1.720090 2.565333 1.979201 3.011815 2.030810 1.419327  
##   
## $predicted$`1`  
## [1] 1.000 2.000 2.000 4.000 2.000 4.000 4.000 1.000 2.000 2.000 2.000 1.000 2.000 1.000 1.000 4.000  
## [17] 4.000 2.000 2.000 2.000 2.000 2.998 1.000 3.998 1.000 3.000 3.998 4.000 2.000 2.000 2.000 3.000  
## [33] 2.000 2.000 2.000 2.000 2.000 2.998 2.000 1.000 2.998 1.000 1.000 2.000 2.000 2.000 4.000 2.000  
## [49] 1.000 2.000 4.000 1.000 2.000 3.000 1.000 1.000 2.996 1.000 4.000 2.000 2.000 1.000 3.000 2.000  
## [65] 4.000 2.000 1.000

Podemos ver el error, bajo la variable $error.cv, y podemos ver las predicciones que se han hecho para cada una de las n.var.

## SVM de Kernel Lineal

Lo bueno que tiene SVM es que es muy robusto frente a la dimensión, por lo que deberíamos de obtener a priori buenos resultados con este método.

Lo primero que hay que hacer es importar la librería…

#install.packages("e1071")  
library("e1071")

## Warning: package 'e1071' was built under R version 3.5.2

Con este método no necesito tener un conjunto de entrenaminento y otro de test, por lo que sigo adelante.

Ahora que hemos instalado la librería, vamos a crear el SVM:

modelo\_svm <- svm(grupo ~ ., data=dataset[2:26], kernel="linear")  
summary(modelo\_svm)

##   
## Call:  
## svm(formula = grupo ~ ., data = dataset[2:26], kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: eps-regression   
## SVM-Kernel: linear   
## cost: 1   
## gamma: 0.04166667   
## epsilon: 0.1   
##   
##   
## Number of Support Vectors: 64

Ahora que tenemos creado este primer modelo, toca predecir:

prediccion <- predict(modelo\_svm,dataset[,2:25])  
prediccion

## 1 2 3 4 5 6 7 8 9 10 11   
## 2.710785 1.167141 2.103294 2.272653 1.921987 2.951614 2.215446 1.903501 2.343303 2.364138 1.624035   
## 12 13 14 15 16 17 18 19 20 21 22   
## 1.264225 1.134063 3.102210 1.472399 3.897730 2.237491 1.897509 2.142401 1.897167 2.651879 1.472399   
## 23 24 25 26 27 28 29 30 31 32 33   
## 1.102637 1.498541 1.595378 2.586326 3.896878 2.912379 1.897073 2.064304 1.392893 2.897015 2.102379   
## 34 35 36 37 38 39 40 41 42 43 44   
## 1.897257 2.712875 1.764464 1.925354 2.268604 1.897013 1.237698 1.952647 1.724457 2.010257 2.102934   
## 45 46 47 48 49 50 51 52 53 54 55   
## 2.102424 2.102300 2.865796 2.103005 2.080632 2.155359 1.844763 2.016170 1.897797 2.896991 2.171207   
## 56 57 58 59 60 61 62 63 64 65 66   
## 2.075540 2.088427 1.103007 2.142982 2.187305 2.102484 2.155098 2.270547 2.102423 3.897624 2.103058   
## 67   
## 2.053638

El problema que tenemos con estas predicciones es que están siendo contínuas, y no discretas, por lo que las voy a discretizar redondeando:

prediccion <- round(prediccion, digits = 0)  
prediccion

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33   
## 3 1 2 2 2 3 2 2 2 2 2 1 1 3 1 4 2 2 2 2 3 1 1 1 2 3 4 3 2 2 1 3 2   
## 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66   
## 2 3 2 2 2 2 1 2 2 2 2 2 2 3 2 2 2 2 2 2 3 2 2 2 1 2 2 2 2 2 2 4 2   
## 67   
## 2

Ahora que hemos predicho, tenemos que sacar la matriz de confusión:

matriz.conf <- table(prediccion, dataset[,26])  
matriz.conf

##   
## prediccion 1 2 3 4  
## 1 5 3 1 1  
## 2 10 25 4 5  
## 3 2 2 3 3  
## 4 0 0 0 3

sum(diag(matriz.conf))/67

## [1] 0.5373134

Obtenemos un porcentaje de acierto medio, pero esto es sin tener en cuenta que los pacientes del grupo 2 pueden pertenecer al 1, lo cual suma alrededor de 10 pacientes más, por lo que obtendríamos valores mucho más altos que rondarían el 65-70% de acierto.

## SVM de Kernel RBF

modelo\_svm.radial <- svm(grupo ~ ., data=dataset[2:26], kernel="radial")  
summary(modelo\_svm.radial)

##   
## Call:  
## svm(formula = grupo ~ ., data = dataset[2:26], kernel = "radial")  
##   
##   
## Parameters:  
## SVM-Type: eps-regression   
## SVM-Kernel: radial   
## cost: 1   
## gamma: 0.04166667   
## epsilon: 0.1   
##   
##   
## Number of Support Vectors: 60

Ahora que tenemos creado este primer modelo, toca predecir:

prediccion.radial <- predict(modelo\_svm.radial,dataset[,2:25])  
prediccion.radial

## 1 2 3 4 5 6 7 8 9 10 11   
## 1.549973 1.897491 2.102992 2.931869 1.958610 2.933492 2.460079 1.968849 2.102839 2.102609 2.102858   
## 12 13 14 15 16 17 18 19 20 21 22   
## 1.834806 1.897158 1.792309 2.081453 3.345387 2.644334 1.896957 2.102818 2.101751 2.102659 2.081453   
## 23 24 25 26 27 28 29 30 31 32 33   
## 1.102789 2.171444 1.688974 2.247323 3.098435 2.310026 2.102645 2.090946 1.897066 2.757770 2.102934   
## 34 35 36 37 38 39 40 41 42 43 44   
## 1.935061 2.103190 1.897447 2.078152 2.427070 1.897235 1.396473 2.724358 1.577114 1.558528 1.911562   
## 45 46 47 48 49 50 51 52 53 54 55   
## 2.102722 2.093837 2.558446 2.102649 1.964595 2.102263 1.784159 1.755280 2.102747 2.897155 1.865630   
## 56 57 58 59 60 61 62 63 64 65 66   
## 1.404453 2.647560 1.341993 2.806776 2.103103 2.102609 1.945673 2.749015 2.102654 3.127086 2.102353   
## 67   
## 1.482761

El problema que tenemos con estas predicciones es que están siendo contínuas, y no discretas, por lo que las voy a discretizar redondeando:

prediccion.radial <- round(prediccion.radial, digits = 0)  
prediccion.radial

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33   
## 2 2 2 3 2 3 2 2 2 2 2 2 2 2 2 3 3 2 2 2 2 2 1 2 2 2 3 2 2 2 2 3 2   
## 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66   
## 2 2 2 2 2 2 1 3 2 2 2 2 2 3 2 2 2 2 2 2 3 2 1 3 1 3 2 2 2 3 2 3 2   
## 67   
## 1

Ahora que hemos predicho, tenemos que sacar la matriz de confusión:

matriz.conf.radial <- table(prediccion.radial, dataset[,26])  
matriz.conf.radial

##   
## prediccion.radial 1 2 3 4  
## 1 5 0 0 0  
## 2 12 30 3 4  
## 3 0 0 5 8

sum(diag(matriz.conf.radial))/67

## [1] 0.5970149

Obtenemos un acierto del 60%, al que hay que sumar otros 12 pacientes. ### Si añadimos estos pacientes, nos encontramos con un acierto del 77% Por lo tanto, SVM de Kernel radial es una buena técnica para la predicción en este problema.

Ahora pasamos a los modelos de inteligencia artificial no supervisados:

# Modelos de inteligencia artificial no supervisados

El primer modelo de inteligencia artificial no supervisado que voy a usar es un modelo de clustering llamado Dendrograma.

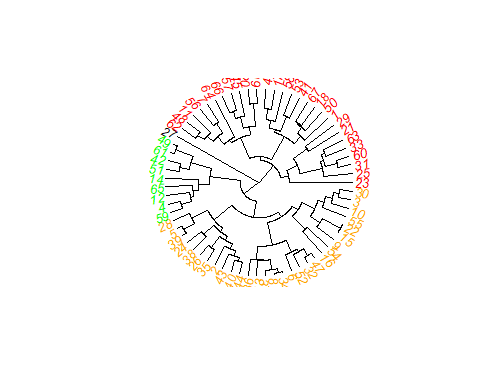
## Dendrograma

Para esto, lo que voy a hacer es dividirlo en 4 clusters, coincidiendo con los 4 grupos de trastornos que tengo.

#install.packages("ape")  
library(ape)

## Warning: package 'ape' was built under R version 3.5.2

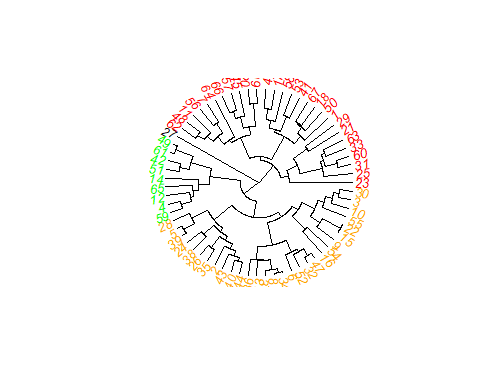
dd <- dist(scale(dataset[,2:25]), method = "euclidean") #Nos basamos en la distancia euclídea  
hier.clust <- hclust(dd, method = "ward.D2")  
colores.dendrograma = c("red", "orange", "green", "black")  
cluster.4 = cutree(hier.clust, 4)  
plot(as.phylo(hier.clust), type = "fan", tip.color = colores.dendrograma[cluster.4], label.offset = 0.3, cex = 0.8)



Como vemos, estamos obteniendo el indentificador de cada paciente en el dendrograma, donde los pacientes que mas se parecen estarán más juntos, mientras que los que menos se parecen estarán más separados. Es interesante analizar como los pacientes verdes y los naranjas surgen de la misma salida del centro, cosa que no ocurre con los rojos y los negros, lo cual quiere decir que algo tienen en común estos dos tipos de casos.

Ahora voy a hacer el mismo dendrograma pero con el DataSet de centrado y escalado, de tal manera que veamos a ver si hay diferencias:

dd <- dist(scale(matriz.pacientes.datos.centscal), method = "euclidean") #Nos basamos en la distancia euclídea  
hier.clust <- hclust(dd, method = "ward.D2")  
colores.dendrograma = c("red", "orange", "green", "black")  
cluster.4 = cutree(hier.clust, 4)  
plot(as.phylo(hier.clust), type = "fan", tip.color = colores.dendrograma[cluster.4], label.offset = 0.3, cex = 0.8)



Si lo comparamos, vemos que hemos obtenido exactamente el mismo resultado, por lo que en este caso el centrado y escalado no es necesario.

Vamos a analizar algunos pacientes aleatoriamente para ver si ha acertado, o al menos si se ha aproximado:

Paciente 1: Analizado - Rojo ——— Real: 1 Paciente 2: Analizado - Naranja —— Real: 2 Paciente 3: Analizado - Naranja —— Real: 2 Paciente 4: Analizado - Verde ——– Real: 4 Paciente 5: Analizado - Naranja —— Real: 2 Paciente 27: Analizado - Negro ——- Real: 4

Es decir, a la vista de estos resultados, podemos concluir que el grupo 1 es el de los pacientes en rojo, el grupo 2 es el de los pacientes en naranja, y luego entre el grupo 3 y el grupo 4 hay dudas, pero teniendo varios pacientes tanto del grupo 3 como del grupo 4 en nuestro DataSet parece ser que algo de error ha cometido.

## K-Means

El algoritmo KMeans en principio no es el algoritmo más adecuado para este trabajo, ya que se basa en círculos para la clasificación de los individuos, cuando en principio en mis datos esto no es así. De todas formas, voy a clasificar a los pacientes siguiendo este algoritmo para comprobar la eficacia que tiene sobre mi problema:

set.seed(76964057) #Seed para reproducibilidad  
k <-kmeans(dataset[,2:25], centers=4) #Creo 4 clusters  
k$centers #Muestro los centros

## edad sex rel\_ctxo\_rel\_mala rel\_ctxo\_trauma rel\_ctxo\_buena ed\_perm ed\_norm  
## 1 16.61111 0.2777778 0.05555556 0.3888889 0.5555556 0.61111111 0.1666667  
## 2 23.82609 0.1739130 0.08695652 0.3043478 0.6086957 0.08695652 0.6956522  
## 3 30.94118 0.1176471 0.17647059 0.3529412 0.4705882 0.11764706 0.6470588  
## 4 44.44444 0.3333333 0.33333333 0.4444444 0.2222222 0.44444444 0.3333333  
## ed\_estr resil\_ba resil\_me resil\_al pen\_dic gen\_ex etiq fil\_men max\_min  
## 1 0.2222222 0.9444444 0.05555556 0.0000000 0.8333333 0.9444444 0.7777778 0.7222222 0.9444444  
## 2 0.2173913 0.4782609 0.52173913 0.0000000 0.9130435 0.9565217 1.0000000 0.7826087 0.9565217  
## 3 0.2352941 0.4705882 0.52941176 0.0000000 0.9411765 1.0000000 0.5882353 0.8235294 1.0000000  
## 4 0.2222222 0.2222222 0.66666667 0.1111111 0.8888889 0.8888889 0.3333333 0.8888889 1.0000000  
## conc\_arb pseu\_res deb raz\_emo inhib asert agres impuls  
## 1 0.9444444 0.2222222 0.7777778 0.8333333 0.6666667 0.00000000 0.3333333 0.5000000  
## 2 1.0000000 0.6956522 1.0000000 0.9130435 0.7826087 0.08695652 0.1304348 0.5652174  
## 3 1.0000000 0.6470588 1.0000000 0.7058824 0.5294118 0.23529412 0.2352941 0.7647059  
## 4 1.0000000 0.3333333 1.0000000 0.5555556 0.5555556 0.33333333 0.1111111 0.6666667

table(k$cluster) #Número de puntos en cada cluster

##   
## 1 2 3 4   
## 18 23 17 9

Interpretando estos resultados, obtenemos:

El cluster 1 destaca por sexo más hacia masculino que otros, una relación contexto ciertamente buena, una educación permisiva, una resiliencia baja, maximización y minimización, razonamiento emocional, cierta inhibición y poca agresividad.

El cluster 2 destaca por una edad mayor, es el cluster con mejor relación con el contexto, y suelen tener las personas de este cluster una educación normal. Destaca por una resiliencia media, pensamiento dicotómico, generalización excesiva, etiquetado, conclusiones arbitrarias, deberías, razonamiento emocional e inhibición.

El cluster número 3 destaca por tener una edad aún más elevada, más ratio de personas del sexo femenino que ningún otro cluster, y tienen una relación con el contexto bastante variable. La educación de estas personas es principalmente normal, con una resiliencia que puede ser tanto baja como media. Destacan por el pensamiento dicotómico, generalización excesiva, poco etiquetado, maximización y minimización, filtro mental, conclusiones arbitrarias, pseudoresponsabilidad, deberías, y suelen ser bastante inhibidos e impulsivos.

Finalmente, el cluster 4 destaca por ser el que tiene la edad más elevada y el ratio de sexo más masculino. La relación con el contexto de estos individuos clasificados en este grupo es principalmente de trauma, aunque también hay buenas y malas. La educación de estos individuos es principalmente permisiva, y la resiliciencia tiende a media. Destacan por la poca etiquetación que hacen, pero un gran fitro mental, conclusiones arbitrarias, poca pseudo-responsabilidad, muchos deberías, poco razonamiento emocional, y son principalmente inhibidos e impulsivos.

Finalmente, podemos ver como ha introducido a 18 individuos en el cluster 1, 23 en el cluster 2, 17 en el cluster 3 y 9 en el cluster 4.