Tarea

Katherine Criollo, Christian Guanoquiza, Esteban Narea

Métodos de clasificación

Veremos un resumen de todos los métodos que hemos visto incluyendo Knn y Naive Bayes. Tened en cuenta que es un método de clasificación multiclase con más de 2 niveles.

Cargamos librerías

```
#Principal
library(ggplot2)
library(ggpubr)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(glmnet) ## regresiones logisitcas
```

```
Loading required package: Matrix

Loaded glmnet 4.1-7
```

```r
library(caret) ### bayes y knn
```

```
Loading required package: lattice
```

```r
library(e1071) ## bayes
```

Cargamos datos

```r
#Principal
# quitamos la primera columna
datos <- read.table("./yeast.data",header = F)[,-1]
```

Creamos las funciones que vamos a necesitar, es decir las funciones de transformación

```r
#Principal
min.max.mean <- function(X) apply(X,2,function(x) (x-mean(x))/(max(x)-min(x)))
min.max.median <- function(X) apply(X,2,function(x) (x-median(x))/(max(x)-min(x)))
min.max <- function(X) apply(X,2,function(x) (x-min(x))/(max(x)-min(x)))
zscore <- function(X) apply(X,2,function(x) (x-mean(x))/sd(x))
l2 <- function(X) apply(X,2,function(x) x/sqrt(sum(x^2)))
```

Para hacer las transformaciones, solamente necesitamos las variables numéricas.

```r
#Principal
datos <- as.data.frame(datos)
datos.numericos <- datos[, which(unlist(lapply(datos, is.numeric)))]
clase <- datos$V10 <- as.factor(datos$V10)
colnames(datos.numericos) <- paste0("Var", rep(1:8))
### procedemos a crear una lista con todas las transformaciones

datos.lista <- list(
  raw = bind_cols(datos.numericos,clase=clase),
  zscore = bind_cols(zscore(datos.numericos),
                     clase = clase),
  l2 = bind_cols(l2(datos.numericos), clase = clase),
  media = bind_cols(min.max.mean(datos.numericos), clase =
                      clase),
  mediana = bind_cols(min.max.median(datos.numericos), clase =
                        clase),
```

```
    min_max = bind_cols(min.max(datos.numericos),
    clase = clase))
```

Descriptiva Gráfica

Al ser demasiadas variables, podemos realizar un `melt`

```
lista_graficos <- vector("list",length=length(datos.lista))
datos.melt <- lapply(datos.lista,reshape2::melt)
```

```
Using clase as id variables
Using clase as id variables
Using clase as id variables
Using clase as id variables
Using clase as id variables
Using clase as id variables
```

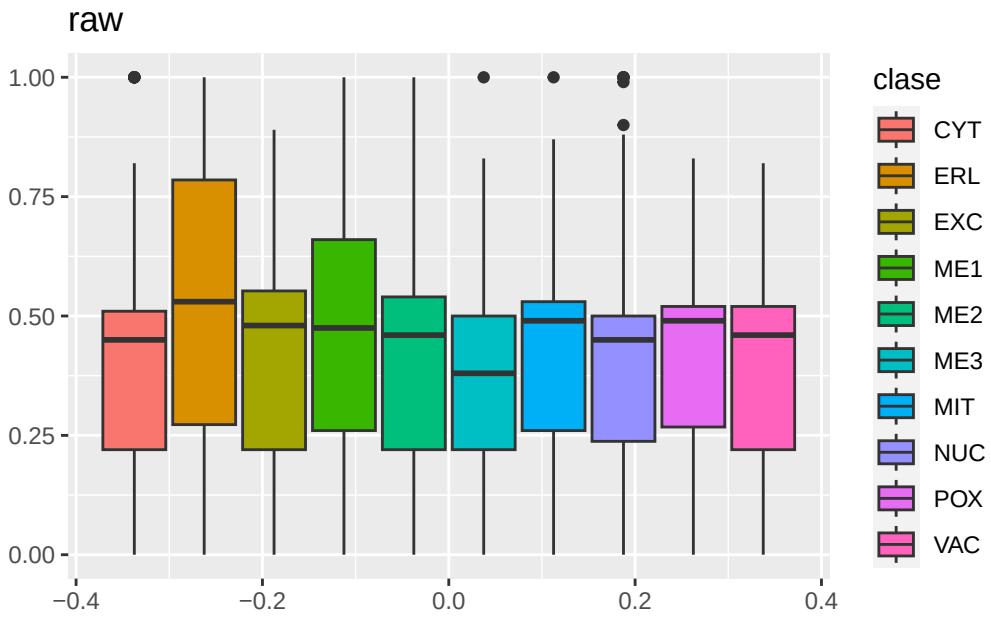
Podemos ver la cabecera de alguna transfomacion para ver el nombre nuevo de las variables

```
head(datos.melt$zscore)
```

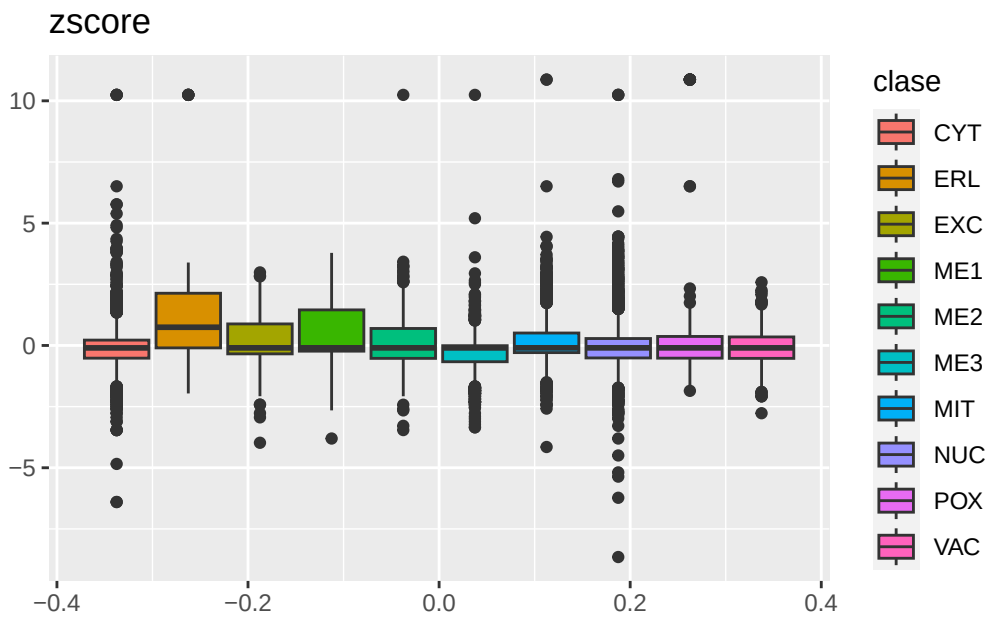
```
  clase variable       value
1   MIT     Var1  0.58178524
2   MIT     Var1 -0.51071851
3   MIT     Var1  1.01878674
4   NUC     Var1  0.58178524
5   MIT     Var1 -0.58355209
6   CYT     Var1  0.07195016
```

```
for(l in 1:length(datos.melt)){

  X <- datos.melt[[l]]
  nombre <- names(datos.melt)[l]
  lista_graficos[[l]] <- ggplot(X,aes(y=value,fill=clase))+geom_boxplot()+ggtitle(nombre)+


}

names(lista_graficos) <- paste0("plot",1:length(datos.lista))
```

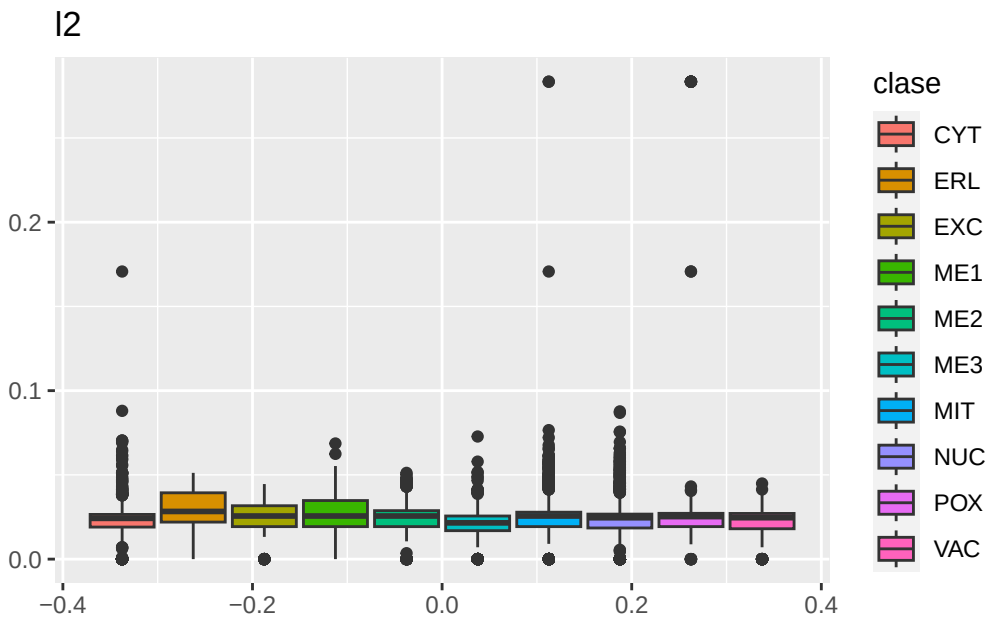
```
lista_graficos$plot1
```



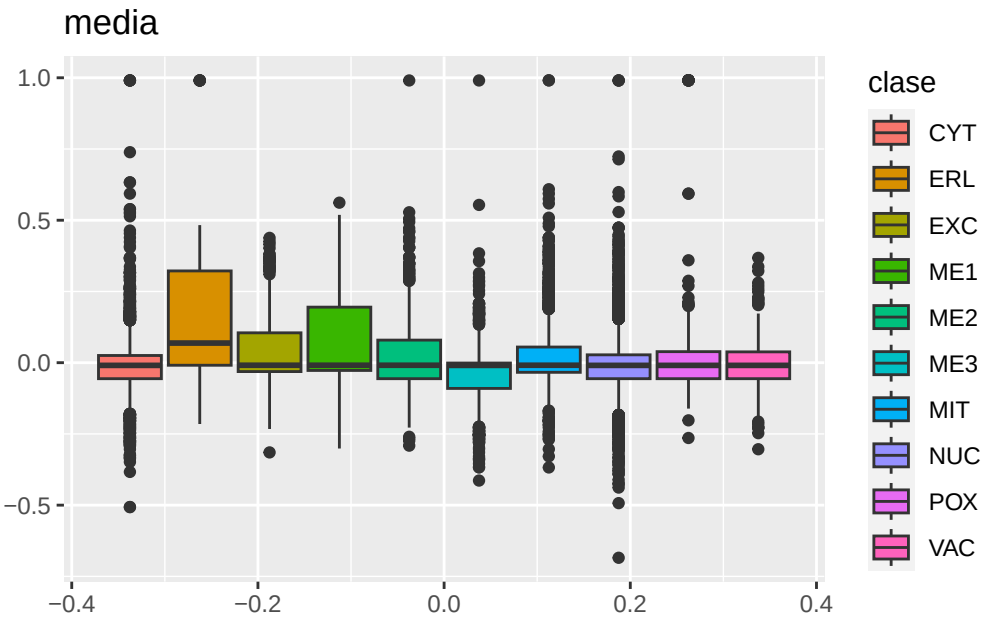

```
lista_graficos$plot2
```



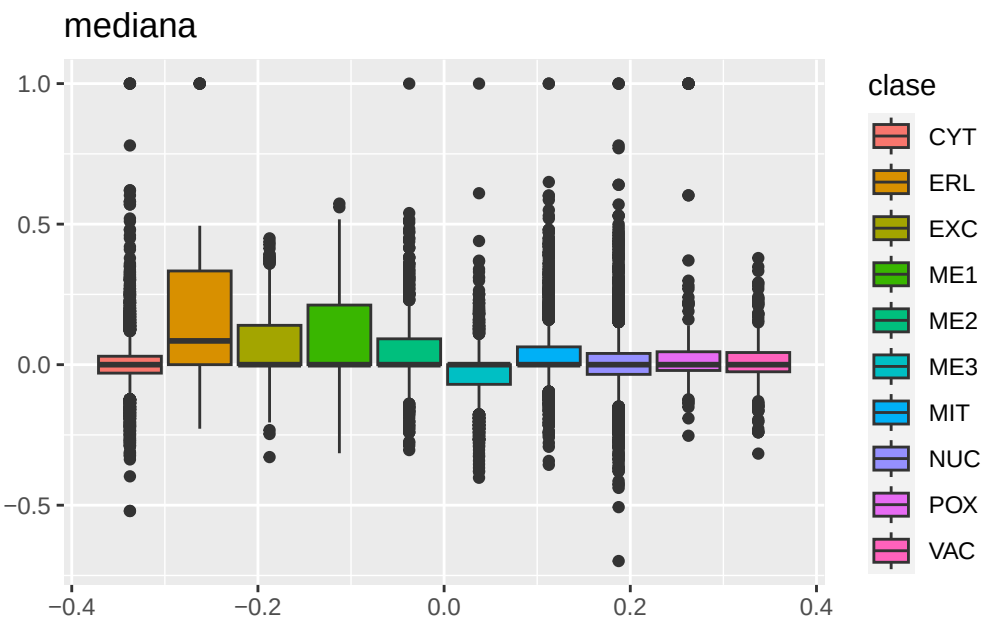

```
lista_graficos$plot3
```



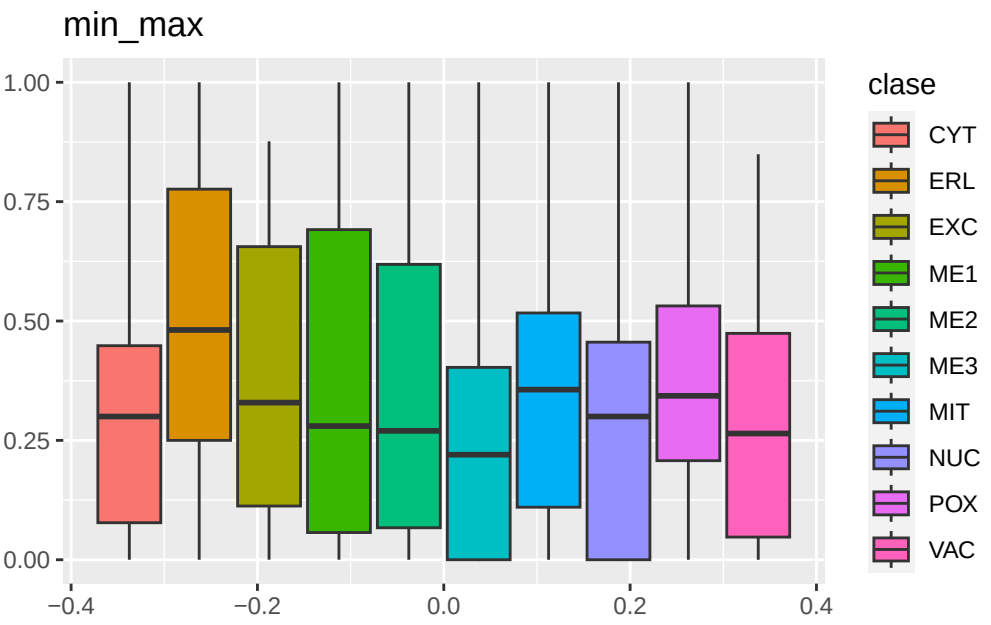
```
lista_graficos$plot4
```




```
lista_graficos$plot5
```



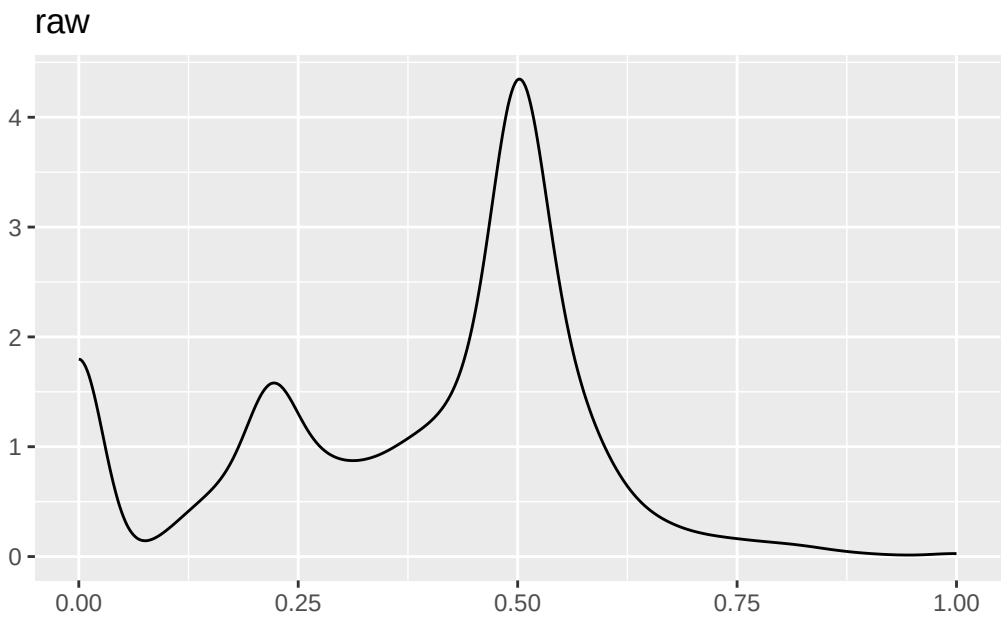

```
lista_graficos$plot6
```



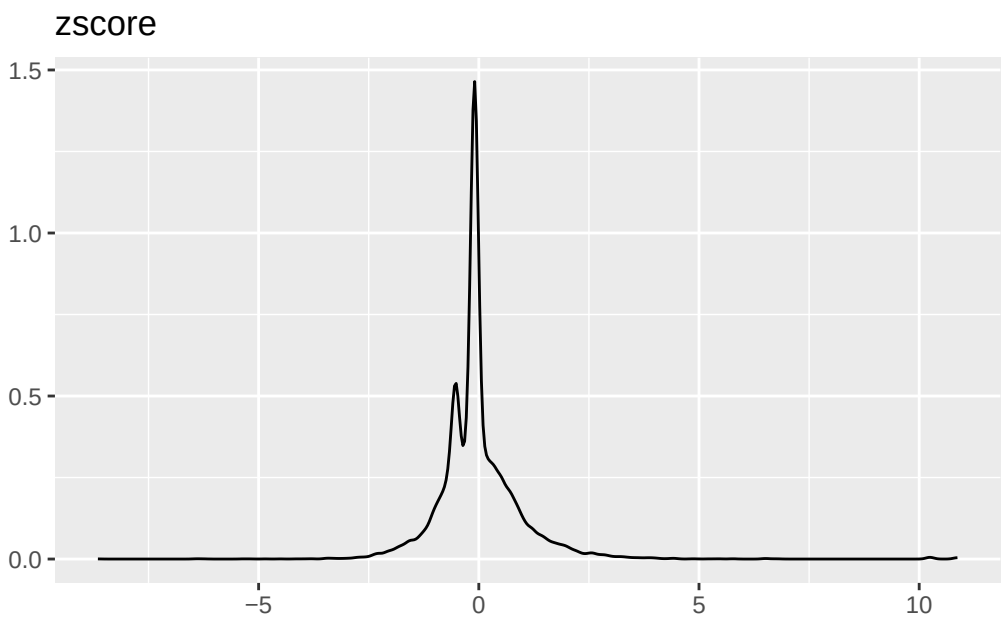
Así por ejemplo la normalización min-max es la mejor, puesto que no tenemos outliers

Otra forma de ver la transfomración es mediante gráficos de densidad

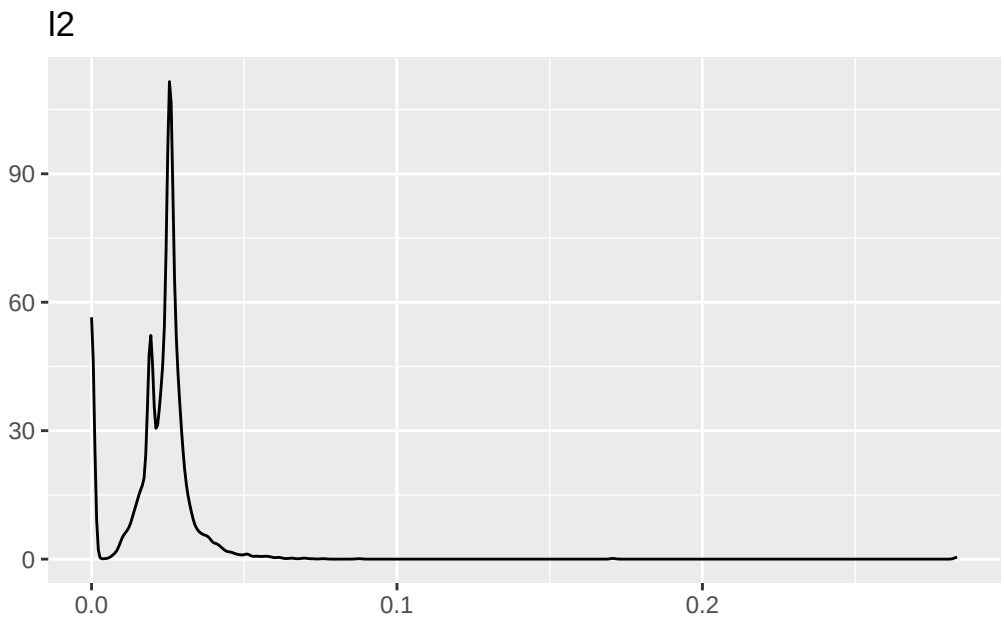
```
for(l in 1:length(datos.melt)){

  X <- datos.melt[[l]]
  nombre <- names(datos.melt)[l]
  lista_graficos[[l]] <- ggplot(X,aes(x=value))+geom_density()+ggtitle(nombre)+xlab("")+yl


}

names(lista_graficos) <- paste0("plot",1:length(datos.lista))

lista_graficos$plot1
```



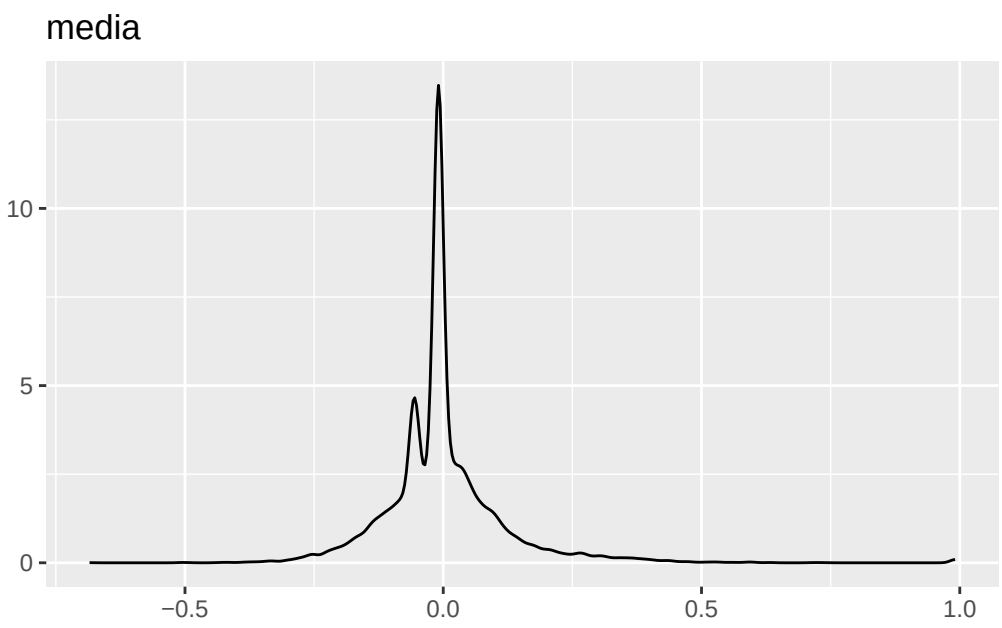

```
lista_graficos$plot2
```



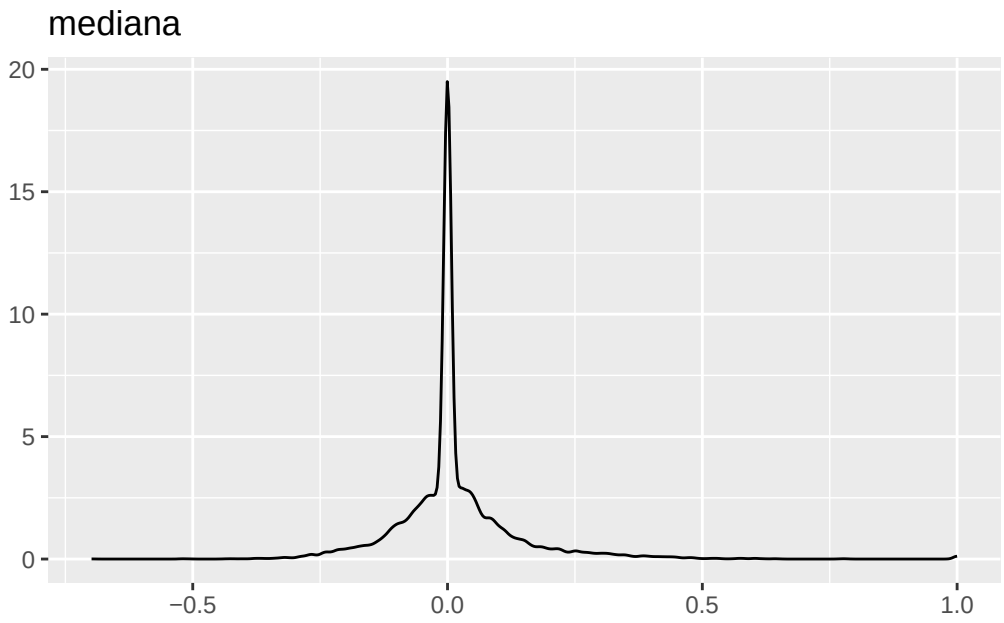

```
lista_graficos$plot3
```



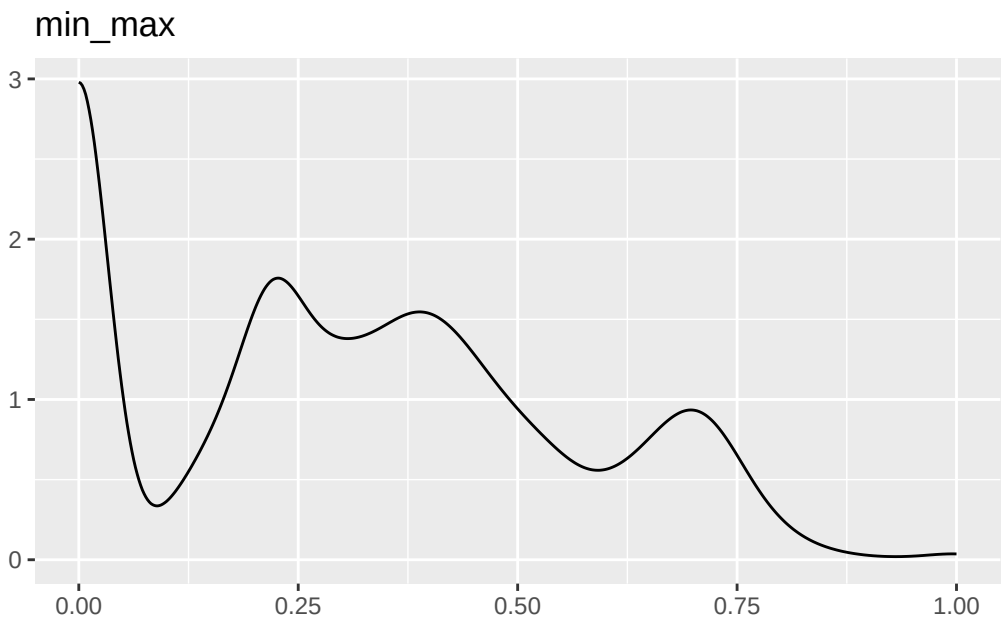
```
lista_graficos$plot4
```




```
lista_graficos$plot5
```

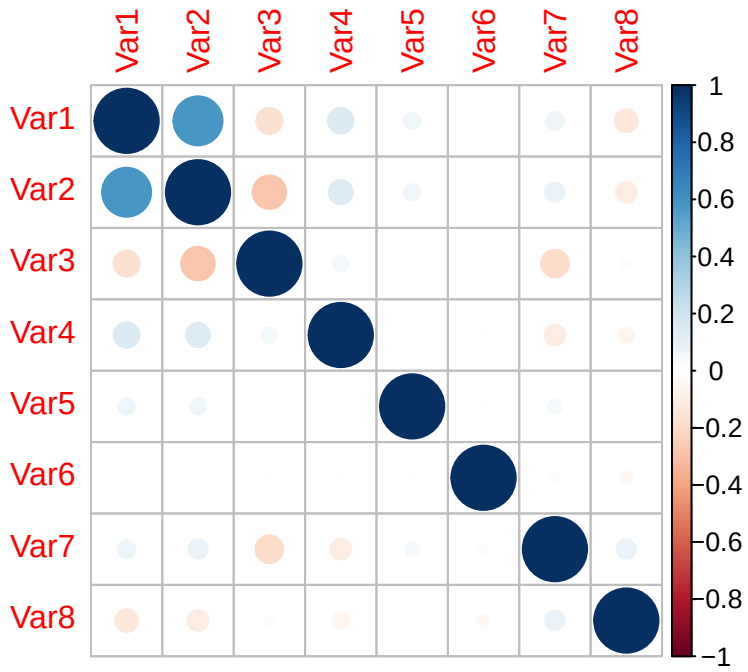



```
lista_graficos$plot6
```

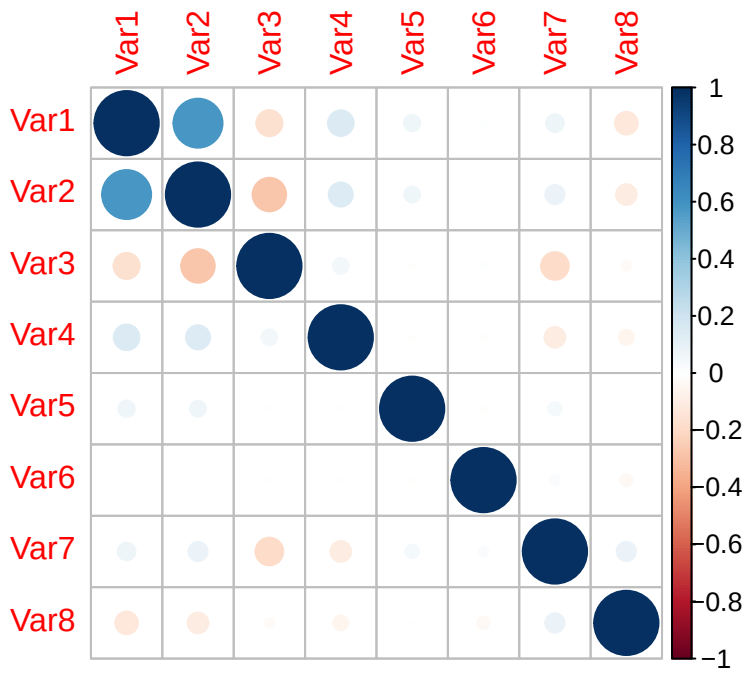


Sin embargo, al ver la densidad, no tenemos una transformacion uniforme.

```
corrplot::corrplot(cor(datos.numericos))
```




```
corrplot::corrplot(cor(datos.lista$media[,-ncol(datos)]))
```



Partición de datos

NOTA: PODEMOS CREAR LA PARTICIÓN CON `caret` o a mano, el 70 porciento de los datos. A mano sería

```
#Principal
set.seed(2796)
n  <- nrow(datos)
idx <- sample(1:n,n*0.7)
### para conjunto de datos podemos realizar el split
datos.train.lista <- lapply(datos.lista, function(x) x[idx,])
datos.test.lista <- lapply(datos.lista, function(x) x[-idx,])
```

Ejemplo regresión logística

alpha=1 es lasso y 0 es ridge

```
#Regresion logistica simple

#Establecemos una semilla para la generación de números aleatorios, se define el esquema d
set.seed(27965)
```

```r
trControl <- trainControl(method = 'cv',
                          number = 10)
myfnlog <- function(x) train(clase ~ ., data = x, method = "multinom", trControl = trContr

# Se aplica una función a cada elemento de una lista y se almacenan los resultados en otra

logistica.lista <- lapply(datos.train.lista,myfnlog)
logisita.pred <- vector("list",length = length(datos.lista))

# Predicciones utilizando el modelo almacenado en cada elemento de logistica.lista
# logisita.pred poseera las prediccionesde a cada modelo en logistica.lista para los datos

for(l in 1:length(datos.lista)){

  logisita.pred[[l]] <- predict(logistica.lista[[l]],datos.test.lista[[l]])

}

# Asignación nombres a los elementos y vector accuracy
names(logisita.pred) <- names(datos.lista)
accuracy <- vector("numeric",length = length(datos.lista))

# Calculo de la precisión para los modelo en la lista logisita.pred, La precisión se almac
for(l in 1:length(datos.lista)){

  accuracy[l] <- confusionMatrix(datos.test.lista$raw$clase,logisita.pred[[l]])$overall[1]

}

# Se asignan nombres a los elementos del vector accuracy basándose en los nombres de la li

names(accuracy) <- names(datos.lista)
print(accuracy)
```

```
      raw    zscore        l2     media   mediana   min_max
0.5739910 0.5762332 0.5627803 0.5739910 0.5739910 0.5739910
```

```r
### Este valor lo tienen que guardar solamente haremos por accuracy y kappa
### tenemos que mirar el objeto matconf
```

```r
# Regresion logistica de lasso

#Establecemos una semilla
#Se define el esquema de validación cruzada y una función para entrenar un modelo de regre
set.seed(27965)
trcontrol <- trainControl(method ='cv', number = 5)
Rlasso <- function(x) train(clase ~ ., data=x, method = "glmnet", trControl = trControl, t

# Se aplica una función a los elemento de la lista y se crea un vector de lista vacío para
lista.regrecion.lasso <- lapply(datos.train.lista, Rlasso)
Prediccion.regrecion.lasso <- vector("list", length = length(datos.lista))

#Predicciones utilizando el modelo almacenado en cada elemento de lista.regresion.lasso
#Prediccion.regrecion.lasso poseera las prediccionesde a cada modelo en lista.regresion la
for(l in 1:length(datos.lista)){
  Prediccion.regrecion.lasso[[l]] <- predict(lista.regrecion.lasso[[l]],datos.test.lista[[
}

names(Prediccion.regrecion.lasso) <- names(datos.lista)
lasso.ac <- vector("numeric", length = length(datos.lista))

# Calculo de la precisión para los modelo en la Prediccion.regresionlasso, La precisión se
for(l in 1: length(datos.lista)){
  lasso.ac[l] <- confusionMatrix(datos.test.lista$raw$clase,Prediccion.regrecion.lasso[[l]

}

names(lasso.ac) <- names(datos.lista)
print(lasso.ac)
```

```
      raw    zscore        l2     media   mediana   min_max
0.5852018 0.5807175 0.5739910 0.5807175 0.5739910 0.5807175
```

```r
# Regresion logistica de Ridge

#Establecemos una semilla
#Se define el esquema de validación cruzada y una función para entrenar un modelo de regre
set.seed(27965)
trcontrol <- trainControl(method ='cv',
                          number = 5)
RRidge <- function(x) train(clase ~ ., data=x, method = "glmnet", trControl = trControl, t
```

```r
# Se aplica una función a los elemento de la lista y se crea un vector de lista vacío para
lista.regrecion.Ridge <- lapply(datos.train.lista, RRidge)
Prediccion.regrecion.Ridge <- vector("list", length = length(datos.lista))

#Predicciones utilizando el modelo almacenado en cada elemento de lista.regresion.Ridge
#Prediccion.regrecion.Ridge poseera las prediccionesde a cada modelo en lista.regresion.Ri
for(l in 1:length(datos.lista)){
  Prediccion.regrecion.Ridge[[l]] <- predict(lista.regrecion.Ridge[[l]],datos.test.lista[[
}

names(Prediccion.regrecion.Ridge) <- names(datos.lista)
Ridge.ac <- vector("numeric", length = length(datos.lista))

# Calculo de la precisión para los modelo, el vector Ridge.ac guarda la precisión
for(l in 1: length(datos.lista)){
  Ridge.ac[l] <- confusionMatrix(datos.test.lista$raw$clase,Prediccion.regrecion.Ridge[[l]
}

names(Ridge.ac) <- names(datos.lista)

print(Ridge.ac)
```

```
      raw    zscore        l2     media   mediana   min_max
0.5739910 0.5695067 0.5717489 0.5739910 0.5695067 0.5695067
```

```r
# Regresion logistica de Bayes

#Establecemos una semilla
#Se define el esquema de validación cruzada con 10 pliegues para el entrenamiento y una fu
set.seed(27965)
trcontrol <- trainControl(method ='cv',
                          number = 10)
RBayes <- function(x) train(clase ~ ., data=x, method = "naive_bayes", trControl = trContr

# Se aplica una función a los elemento de la lista y se crea un vector de lista vacío para
lista.regrecion.Bayes <- lapply(datos.train.lista, RBayes)
Prediccion.regrecion.Bayes <- vector("list", length = length(datos.lista))

#Predicciones utilizando el modelo almacenado en cada elemento de lista.regresion.Bayes
#Prediccion.regrecion.Bayes poseera las predicciones de a cada modelo en lista.regresion.B
```

```r
for(l in 1:length(datos.lista)){
  Prediccion.regrecion.Bayes[[l]] <- predict(lista.regrecion.Bayes[[l]],datos.test.lista[[
}

names(Prediccion.regrecion.Bayes) <- names(datos.lista)
Bayes.ac <- vector("numeric", length = length(datos.lista))

# Calculo de la precisión para los modelo, el vector Bayes.ac guarda la precisión
for(l in 1: length(datos.lista)){
  Bayes.ac[l] <- confusionMatrix(datos.test.lista$raw$clase,Prediccion.regrecion.Bayes[[l]
}

names(Bayes.ac) <- names(datos.lista)
print(Bayes.ac)
```

```
      raw    zscore        l2     media   mediana   min_max
0.3744395 0.5156951 0.4439462 0.5156951 0.3744395 0.3744395
```

```r
# Regresion logistica Knn

#Establecemos una semilla
#Se define el esquema de validación cruzada con 10 pliegues para el entrenamiento y una fu
set.seed(27965)

VKnn = c(1:20)

trcontrol <- trainControl(method ='repeatedcv', number = 3, repeats = 10)
RKnn <- function(x) train(clase ~ ., data=x, method = "knn",trControl = trControl, tuneGri

# Se aplica una función a los elemento de la lista y se crea un vector de lista vacío para
lista.regrecion.Knn <- lapply(datos.train.lista, RKnn)
Prediccion.regrecion.Knn <- vector("list", length = length(datos.lista))

#Predicciones utilizando el modelo almacenado en cada elemento de lista.regresion.Knn
#Prediccion.regrecion.Ridge poseera las predicciones de a cada modelo en lista.regresion.K
for(l in 1:length(datos.lista)){
  Prediccion.regrecion.Knn[[l]] <- predict(lista.regrecion.Knn[[l]],datos.test.lista[[l]])
}

names(Prediccion.regrecion.Knn) <- names(datos.lista)
Knn.ac <- vector("numeric", length = length(datos.lista))
```

```
# Calculo de la precisión para los modelo, el vector Ridge.ac guarda la precisión
for(l in 1: length(datos.lista)){
  Knn.ac[l] <- confusionMatrix(datos.test.lista$raw$clase,Prediccion.regrecion.Knn[[l]])$o
}

names(Knn.ac) <- names(datos.lista)
print(Knn.ac)
```

```
      raw    zscore        l2     media   mediana   min_max
0.5695067 0.5739910 0.5672646 0.5807175 0.5919283 0.5695067
```

```
#Obtenemos una matriz 5x6 de los resusltados obtenidos en las regreciones
matriz3d <- matrix(c(accuracy, lasso.ac, Ridge.ac, Bayes.ac, Knn.ac), nrow = 5, ncol = 6,

matriz3d
```

```
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.5739910 0.5762332 0.5627803 0.5739910 0.5739910 0.5739910
[2,] 0.5852018 0.5807175 0.5739910 0.5807175 0.5739910 0.5807175
[3,] 0.5739910 0.5695067 0.5717489 0.5739910 0.5695067 0.5695067
[4,] 0.3744395 0.5156951 0.4439462 0.5156951 0.3744395 0.3744395
[5,] 0.5695067 0.5739910 0.5672646 0.5807175 0.5919283 0.5695067
```

```
# separamos la fila con los resultados mayores
fila_max <- matriz3d[which.max(max.col(matriz3d)), ]
print(fila_max)
```

```
[1] 0.5695067 0.5739910 0.5672646 0.5807175 0.5919283 0.5695067
```