Sistemas de Información y Telemedicina. *

Marta Girones Sanguesa
Silvia Marset Gomis
Ignacio Amat Hernández
Sofía Gutiérrez Santamaría

January 17, 2020

Contents

List of Figures

Listings

*Grado en Ingeniería Biomédica, Escuela Técnica Superior de Ingenieros Industriales, Valencia, España.

1 Resumen

El principal objetivo de este trabajo es adentrarnos en otros lenguajes de programación como son *Python* y *R*, al mismo tiempo que profundizamos en las herramientas para el soporte a la decisión médica. Otro objetivo del trabajo es poder ayudar a otras personas que vayan a realizar un trabajo similar a elegir el lenguaje de programación, mediante una comparación de los tres mencionados anteriormente, que más se ajusta a sus necesidades.

Para lograr una buena comparación de lenguajes nos basaremos en ciertos aspectos como son la facilidad de aprendizaje del lenguaje, la facilidad de comprensión de la asignatura con cada lenguaje, las funciones integradas, incluso los distintos tipos de gráficos que nos ofrece cada uno.

Palabras Clave: Matlab, Python, R, machine learning.

2 Introducción

Debido al aumento de información que se genera y almacena hoy en día es necesario, más que nunca, una clasificación de datos. Los CDSSs basados en datos utilizan técnicas de aprendizaje automático que permiten al sistema aprender de las experiencias pasadas y/o encontrar patrones en los datos clínicos. Estos permiten minimizar el riesgo de fuga de datos, además de ser útil en la ayuda al diagnóstico médico.

Por ello, en el presente trabajo se lleva a cabo un análisis exploratorio de un conjunto de datos de cáncer de mama de Coimbra que incluye las características clínicas de sujetos sanos y enfermos, una selección y extracción de características del conjunto de datos, distintos modelos de clasificación y, por último, diferentes estrategias y métricas de evaluación.

En el análisis exploratorio se realiza un preprocesado de los datos donde se verifica la integridad de la base de datos, se eliminan las filas corruptas y se estudian los datos anómalos. Posteriormente se analiza la correlación entre variables y entre datos, la normalidad de estos datos y su distribución mediante métricas básicas como histograma, kernel density, qqplot y boxplot.

En el proceso de selección y extracción de características investigamos cuáles de las variables son estadísticamente relevantes. Ello se consigue gracias a métodos de selección –aquellos basados en Filters o Wrappers– y de extracción de características –como son Análisis de Componentes Principales y Análisis Discriminante Lineal. Estos métodos tienen como objetivo final la reducción de dimensionalidad.

En cuanto a los modelos de clasificación podemos distinguir los de aprendizaje supervisado, aplicados en este trabajo, y los basados en el aprendizaje no supervisado. Los modelos de clasificación supervisados están basados en el aprendizaje mediante pares dato-solución, como KNN, LDA y QDA. Los métodos no supervisados, en

cambio, son aquellos en los que los sistemas se adaptan mediante las características intrínsecas de los datos.

Por último, para estudiar las estrategias y métricas de evaluación emplearemos la matriz de confusión de datos, la curva ROC y el error cuadrático medio. Estas cuatro fases se realizan en tres lenguajes de programación distintos: Matlab, Python y R.

3 Materiales y Métodos

La base de datos utilizada en este trabajo es un conjunto de datos de cáncer de mama de Coimbra que incluye las características clínicas de 64 pacientes con cáncer de mama y 52 controles sanos. Está formada por 9 predictores –datos antropométricos y parámetros recopilados de análisis de sangre rutinarios– y 1 variable dependiente binaria que indica la presencia o no de cáncer de mama (1: controles saludables y 2: pacientes). Los 9 predictores son: Edad (años), IMC (kg / m2), Glucosa (mg / dL), Insulina (μU / mL), HOMA, Leptina (ng / mL), Adiponectina (μg / mL), Resistencia (ng / mL) y MCP-1 (pg / dL).

Los lenguajes de programación utilizados son Matlab, Python y R. Matlab es un lenguaje interpretado propiedad de MathWorks; requiere de licencia comercial, es ampliamente usado en contextos ingenieriles. Python es un proyecto de código abierto de Guido van Rossum, es un lenguaje de scripting de uso general, no especializado. R es el único lenguaje especializado en ciencia de datos, es el antecesor a S; también de código abierto. Una vez definidos los materiales, procedemos a describir la metodología a seguir; será desarrollada en Python y R. Haremos una comparación con el desarrollo de Matlab realizado a lo largo de la asignatura.

1. Análisis exploratorio:
    a) Cargar datos.
    b) Solucionar existencia de casos perdidos.
    c) Obtener de cada variable númerica y separado por clases el histograma, kernel density, qqplot y boxplot.
    d) Realizar plot matriz de gráficos de dispersión y detección de variables correlacionadas.
    e) Detección y solución de la presencia de outliers.
2. Extracción de características:
    a) Aplicar Filters (fscore y relieff) y Wrappers para seleccionar las variables más significativas.
    b) Normalizar los datos mediante zscore.
    c) Aplicar el Análisis de Componentes Principales para obtener un nuevo conjunto de características relevantes (PCA).
    d) Representar la variabilidad de los datos mediante pareto y la influencia de cada una de las componentes originales para la generación de cada componente principal con biplot.

e) Transformar los datos usando la matriz de variables originales y la matriz de coeficientes que ofrece PCA.

f) Representar las proyecciones de las primeras componentes principales mediante el gráfico de dispersión scatter.

g) Aplicar Análisis Lineal Discriminante (LDA) para obtener un nuevo conjunto de características relevantes.

3. Modelos de clasificación:

a) Separar los datos en una muestra de entrenamiento y otra de validación del modelo a partir del método 'HoldOut'.

b) Entrenar un modelo de clasificación lineal, cuadrático y basado en k-vecinos con distinto número de vecinos.

4. Estrategia y métricas de evaluación:

a) Separar los datos en una muestra de entrenamiento y otra de validación del modelo a partir del método 'KFold'.

b) Entrenar un modelo de clasificación basado en k-vecinos.

c) Obtener el error cuadrático medio, la curva ROC y la matriz de confusión.

4 Resultados

4.1 Análisis exploratorio

4.1.1 Preparación

```python
import numpy as np
from   scipy import stats

#  names of variables
labels = ['age', 'leptin', 'bmi', 'adiponectin', 'glucose',
        'resistin', 'insulin', 'MCP1', 'HOMA']

#  loads data
data = np.loadtxt (open (r'../../data.csv', 'rb'), delimiter = ',', skiprows = 1)

#  rewrites data as all the rows of data w/out nan cells
data = data [~np.isnan (data).any (axis=1)]

#  separates parameters into matrix x
x    = np.array ([list (data [x][:-1])    for x in range (len (data))])

#  and class (1, 2) into vector y
y    = np.array ([int  (data [x][ -1])    for x in range (len (data))])

#  removes outliers
data_no = data [(np.abs (stats.zscore (data)) < 3).all (axis = 1)]
#    ↑ = No Outliers

x_no = np.array ([list (data_no [x][:-1]) for x in range (len (data_no))])
y_no = np.array ([int  (data_no [x][ -1]) for x in range (len (data_no))])
```

Listing 1: Importaciones iniciales y preparacion de datos en Python.

```r
# load data
datos <- read.table ('../../data.csv', sep = ',', header = T)
datos <- na.omit (datos)

# ignore rows w/ components above the 99^th percentile
suppressPackageStartupMessages (library (dplyr))
datos <- datos %>% filter_all (all_vars (. <= quantile (., 0.99, na.rm = T)))
```

Listing 2: Importaciones iniciales y preparacion de datos en R.

```
load ('Data.mat')
% Eliminamos filas que contengan NaN:
dataR2mat = dataR2mat (~any (ismissing (dataR2), 2), :);
% Cuando queramos eliminar datos anomalos usaremos:
dataR2mat = dataR2mat (~any (isoutlier (dataR2mat (:,1:end-1)), 2), :);
```

Listing 3: Importaciones iniciales y preparacion de datos en Matlab.

4.1.2 Histogramas

En este apartado dibujamos los histogramas comparativos.

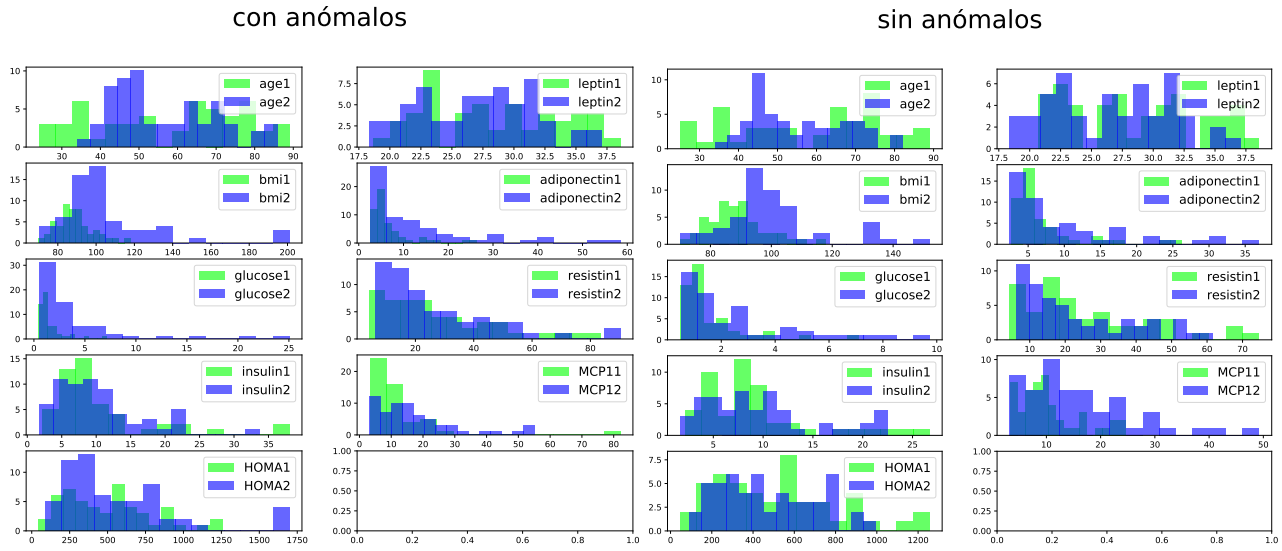

Fig. 1: Histogramas Python para datos con y sin anomalias.

```
import matplotlib as mpl
import matplotlib.pyplot as plt

#  load preprocessed data, x and y are raw, x_no and y_no contain no outliers
from preprocessing import x, y, x_no, y_no, labels

#  colours for the histograms
fc = [(), (0, 1, 0, 0.6), (0, 0, 1, 0.6)]
#          (R, G, B,  α )← transparency

fig, ax = plt.subplots (nrows = 5, ncols = 2, figsize = (13, 10))
ax = ax.flatten ()

#  draws each of the histograms, two for each variable
for i in range (0, 9):
    for j in [1, 2]:
        ax[i].hist (x [y == j, i], bins = 15, fc = fc [j], label = labels [i] + str
            (j))
        ax[i].legend (loc = 1, prop={'size': 15})

fig.suptitle ('con anómalos', fontsize = 30)
fig.savefig  ('../images/hist.pdf', bbox_inches = 'tight', pad_inches = 0)
```

Listing 4: Código Python generador de los histogramas con datos anómalos.

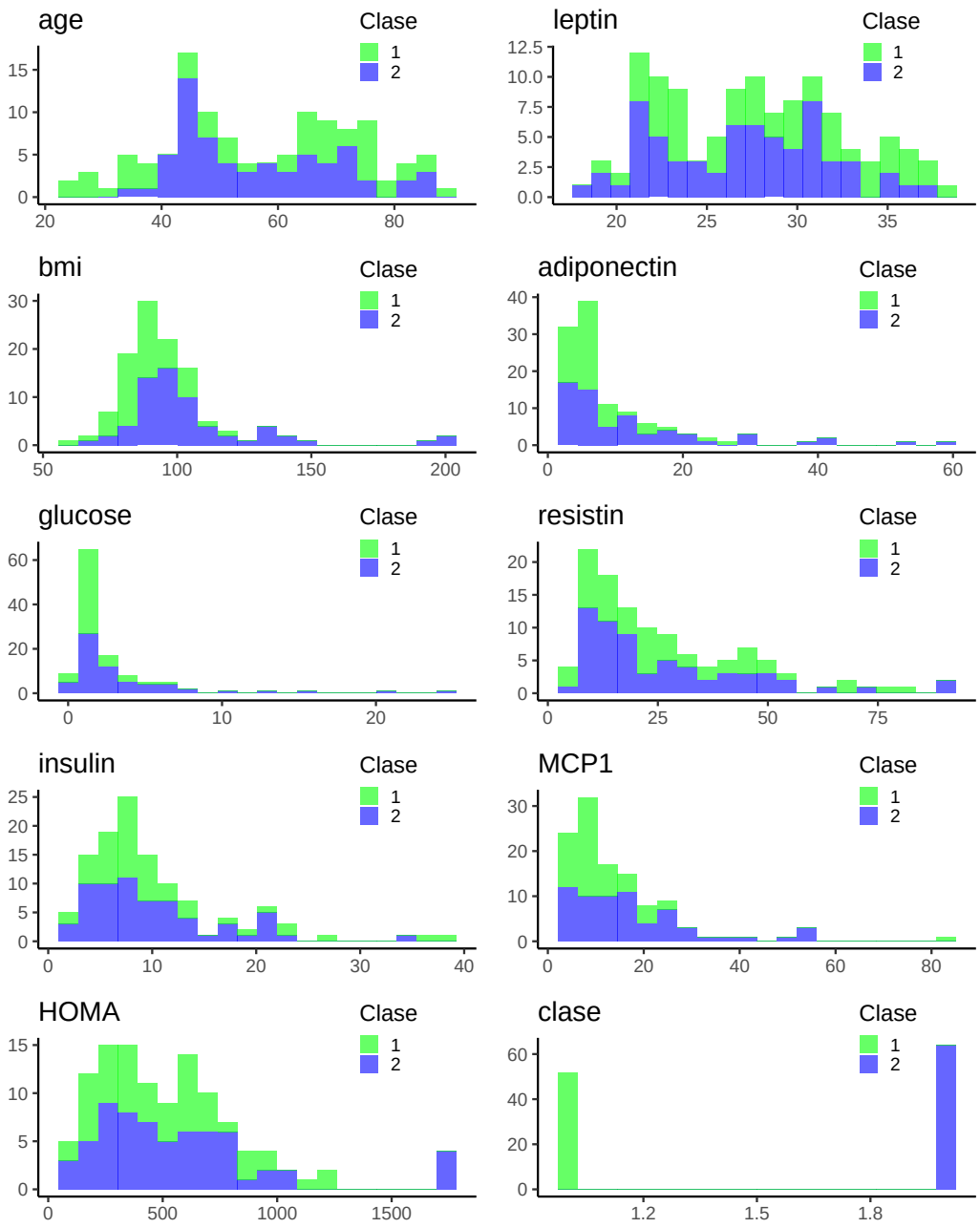

Fig. 2: Histogramas R para datos con anomalias.

```
for (i in 1:10){
pdf (file = paste ('../images/hist', i, '.pdf', sep = ''), width = 6, height = 3)
print (ggplot (datos, aes (x = datos[,i], fill = as.factor (clase))) +
                labs (x = NULL, y = NULL, title = names (datos)[i], fill = 'Clase') +
                geom_histogram  (bins = 20, alpha = 0.6) +
                theme_classic   (base_size = 20) +
                scale_fill_manual(values = c ('green', 'blue')) +
                theme   (legend.position = c (0.8, 1)))
dev.off ()
}
```

Listing 5: Código R generador de los histrogramas con datos anómalos.

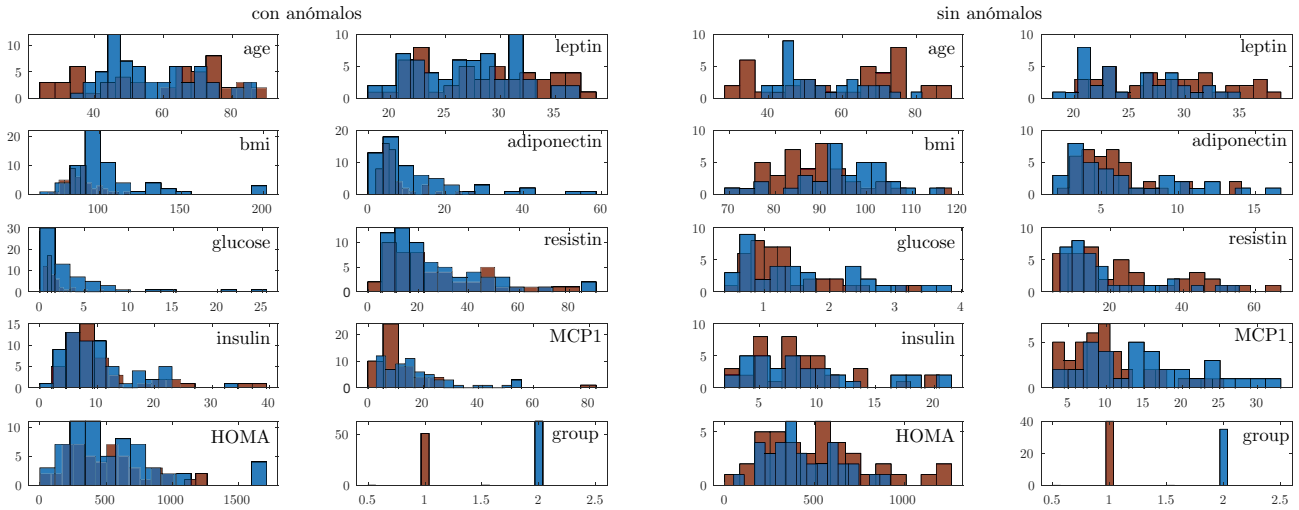

Fig. 3: Histogramas Matlab.

```
for (i = 1 : 10)
        subplot (5, 2, i);
        histogram (dataR2mat (dataR2mat (:, end) == 1, i), 15);
        hold on;
        histogram (dataR2mat (dataR2mat (:, end) == 2, i), 15);
end
```

Listing 6: Código Matlab generador de los histogramas.

4.1.3 Kernel Density

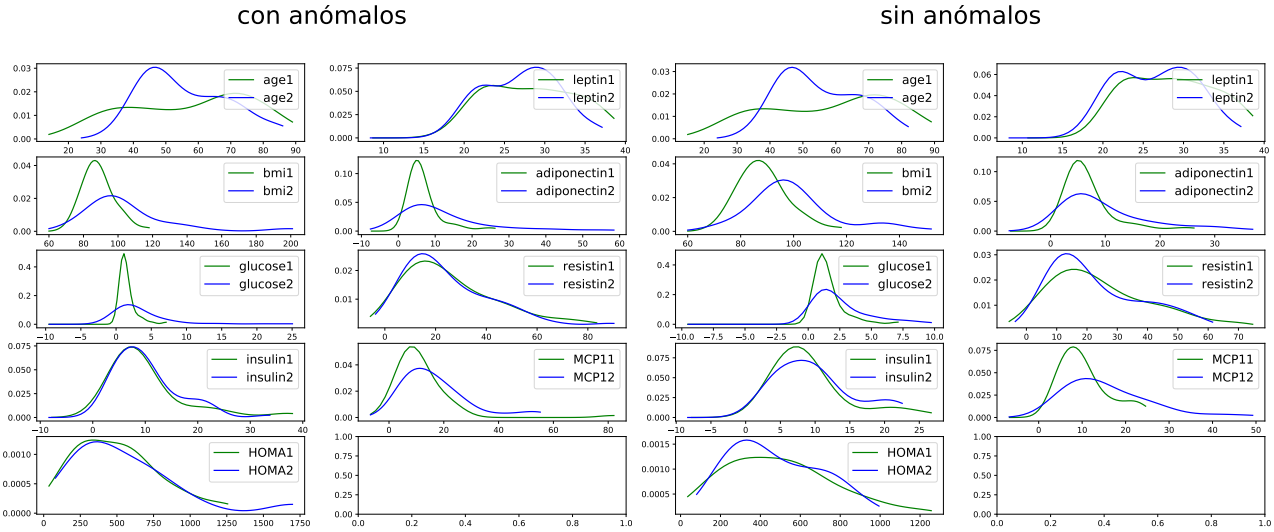

Fig. 4: Kernel Density para datos con y sin anomalias.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import gaussian_kde

#  load preprocessed data, x and y are raw, x_no and y_no contain no outliers
from preprocessing import x, y, x_no, y_no, labels

#  colours
fc = ['', 'green', 'blue']

fig, ax = plt.subplots (nrows = 5, ncols = 2, figsize = (13, 10))
ax = ax.flatten ()

#  same loop in principle as before
for i in range (0, 9):
    for j in [1, 2]:
        kde = gaussian_kde (x_ := x [y == j, i])
        xs = np.linspace(np.min (x_) - 10, np.max (x_), num=len (x_))
        ax[i].plot (xs, kde(xs), c = fc[j], label = labels [i] + str (j))
        ax[i].legend (loc = 1, prop={'size': 15})

fig.suptitle ('con anómalos', fontsize = 30)
fig.savefig  ('../images/kden.pdf', bbox_inches = 'tight', pad_inches = 0)
```

Listing 7: Código Python generador de los kernel density plots con datos anómalos.

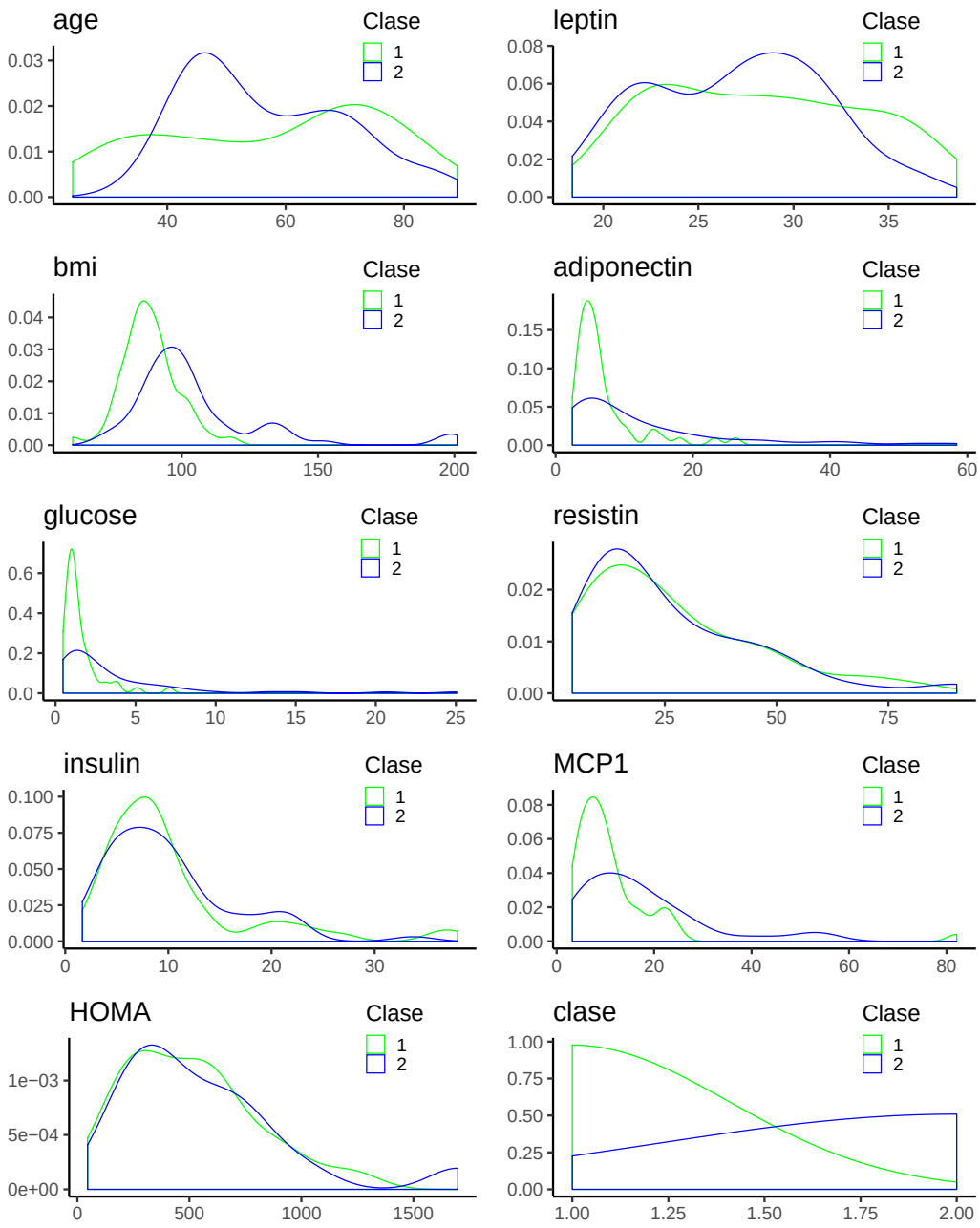

Fig. 5: Gráficos de densidad R.

```
for (i in 1:10){
pdf (file = paste ('../images/dens', i, '.pdf', sep = ''), width = 6, height = 3)
print (ggplot (datos, aes (x = datos[,i], colour = as.factor (clase))) +
                labs (x = NULL, y = NULL,
                      title = names (datos)[i], colour = 'Clase') +
                geom_density () + theme_classic (base_size = 20) +
                scale_colour_manual (values = c ('green', 'blue')) +
                theme   (legend.position = c (0.8, 1)))
dev.off ()
}
```

Listing 8: Código R generador de los density plots.

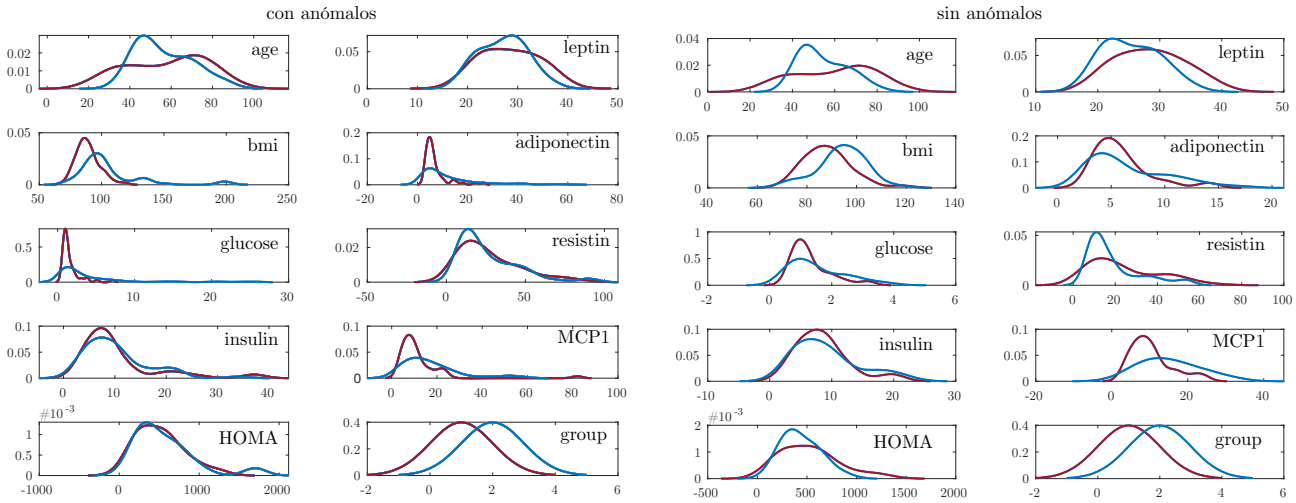

Fig. 6: Gráficos de densidad Matlab.

```
for (i = 1 : 10)
        subplot (5, 2, i);
        ksdensity (dataR2mat (dataR2mat (:, end) == 1, i));
        hold on;
        ksdensity (dataR2mat (dataR2mat (:, end) == 2, i));
end
```

Listing 9: Código Matlab generador de los kdensity.

4.1.4 Boxplot

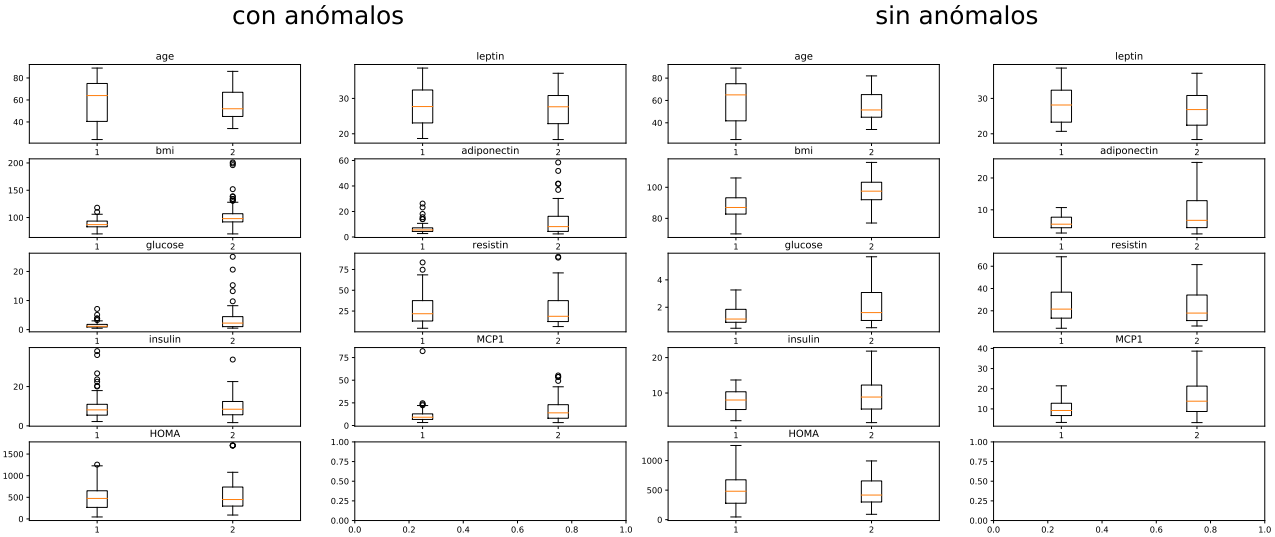

Fig. 7: Boxplots Python para datos con y sin anomalias.

```python
import matplotlib as mpl
import matplotlib.pyplot as plt

#  load preprocessed data, x and y are raw, x_no and y_no contain no outliers
from preprocessing import x, y, x_no, y_no, labels

fig, ax = plt.subplots (nrows = 5, ncols = 2, figsize = (13, 10))
ax = ax.flatten ()

for i in range (0, 9):
    ax[i].boxplot ([x [y == 1, i], x [y == 2, i]])
    ax[i].title.set_text (labels [i])

fig.suptitle ('con anómalos', fontsize = 30)
fig.savefig  ('../images/boxp.pdf', bbox_inches = 'tight', pad_inches = 0)
```

Listing 10: Código Python generador de los boxplots con datos anómalos.

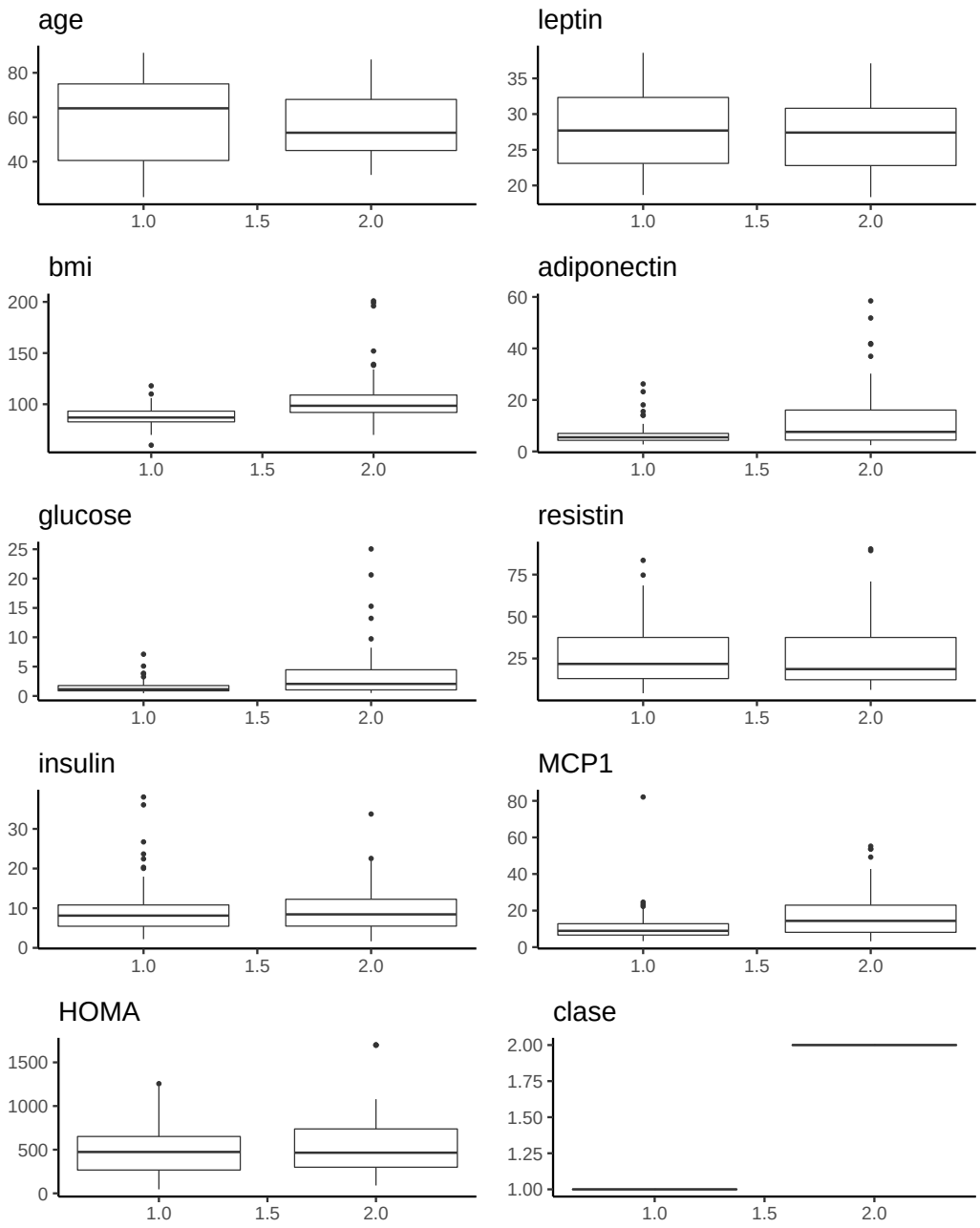

Fig. 8: Boxplots R para datos con anomalias.

```
for (i in 1:10){
pdf (file = paste ('../images/box', i, '.pdf', sep = ''), width = 6, height = 3)
print (ggplot (datos, aes (x = clase,
                           y = datos[,i],
                           group = clase)) +
               labs (x = NULL, y = NULL, title = names (datos)[i]) +
               geom_boxplot  () +
               theme_classic (base_size = 20))
dev.off ()
}
```

Listing 11: Código R generador de los boxplots.

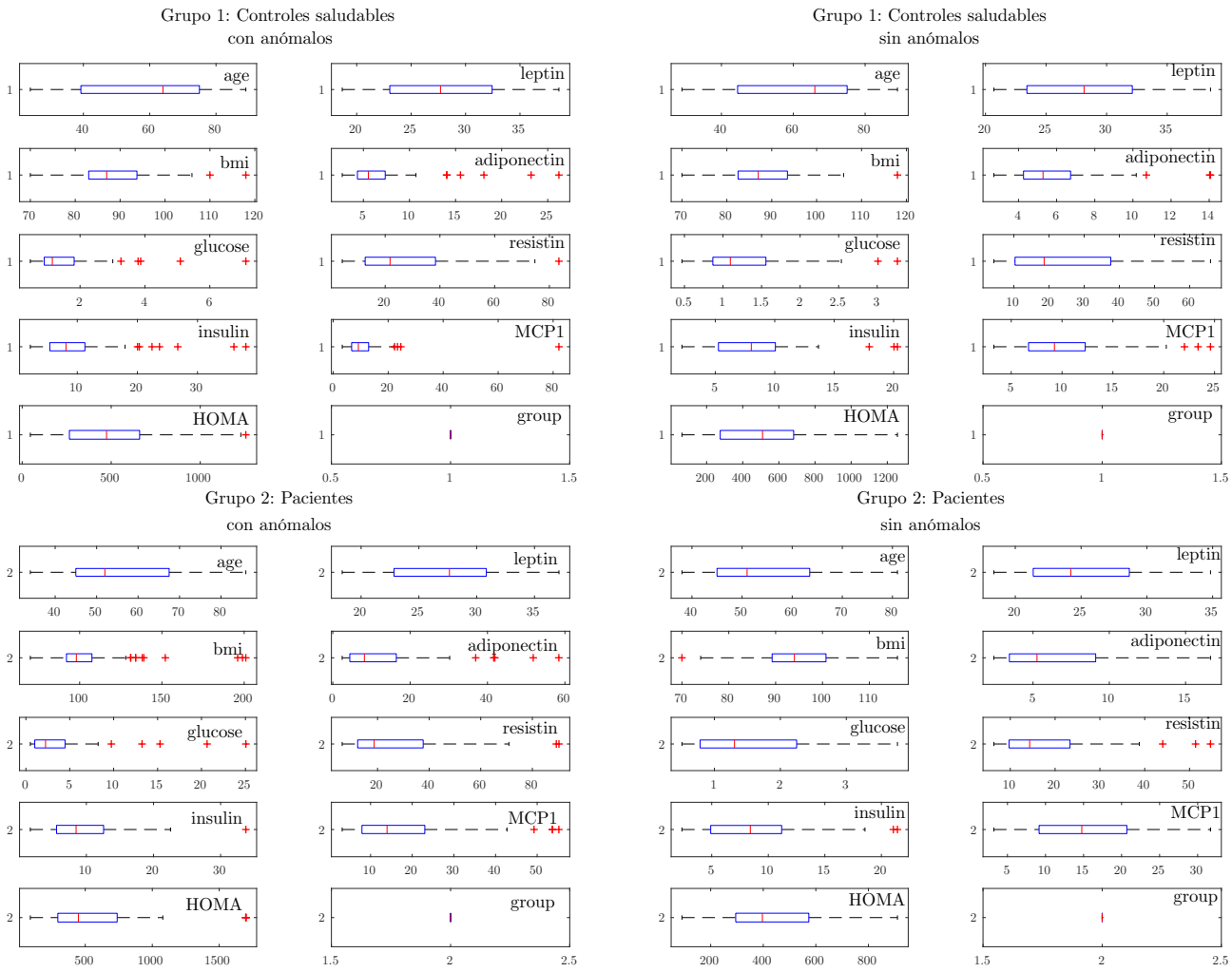

Fig. 9: Boxplots Matlab para datos con anomalias.

```
for (i = 1 : 10)
        subplot (5, 2, i);
        boxplot (dataR2mat (dataR2mat (:, end) == 1, i));
        hold on;
        boxplot (dataR2mat (dataR2mat (:, end) == 2, i));
end
```

Listing 12: Código Matlab generador de los boxplots.

4.1.5 QQplot

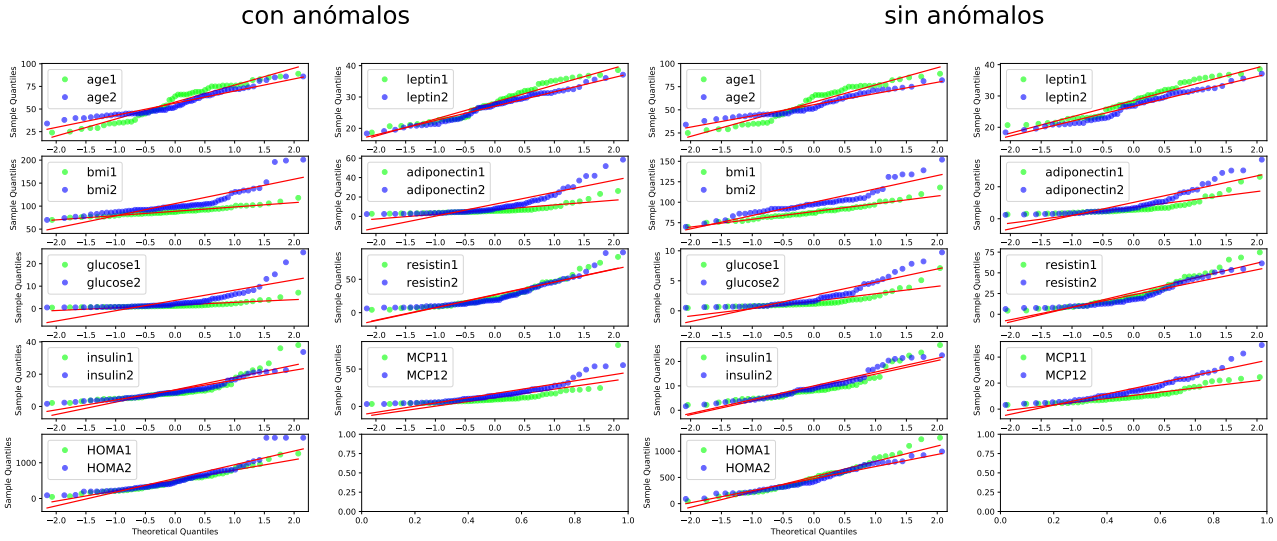

Fig. 10: QQplots Python para datos con y sin anomalias.

```
import matplotlib as mpl
import matplotlib.pyplot as plt

#  load preprocessed data, x and y are raw, x_no and y_no contain no outliers
from preprocessing import x, y, x_no, y_no, labels
import statsmodels.api as sm

fc = [(), (0, 1, 0, 0.6), (0, 0, 1, 0.6)]

fig, ax = plt.subplots (nrows = 5, ncols = 2, figsize = (13, 10))
ax = ax.flatten ()

for i in range (0, 9):
    for j in [1, 2]:
        sm.qqplot (x [y == j, i], ax = ax[i], c = fc[j],
                line = 's', label = labels [i] + str (j))
        ax[i].legend (loc = 2, prop={'size': 15})

fig.suptitle ('con anómalos', fontsize = 30)
fig.savefig  ('../images/qqp.pdf', bbox_inches = 'tight', pad_inches = 0)
```

Listing 13: Código Python generador de los QQplots con datos anómalos.

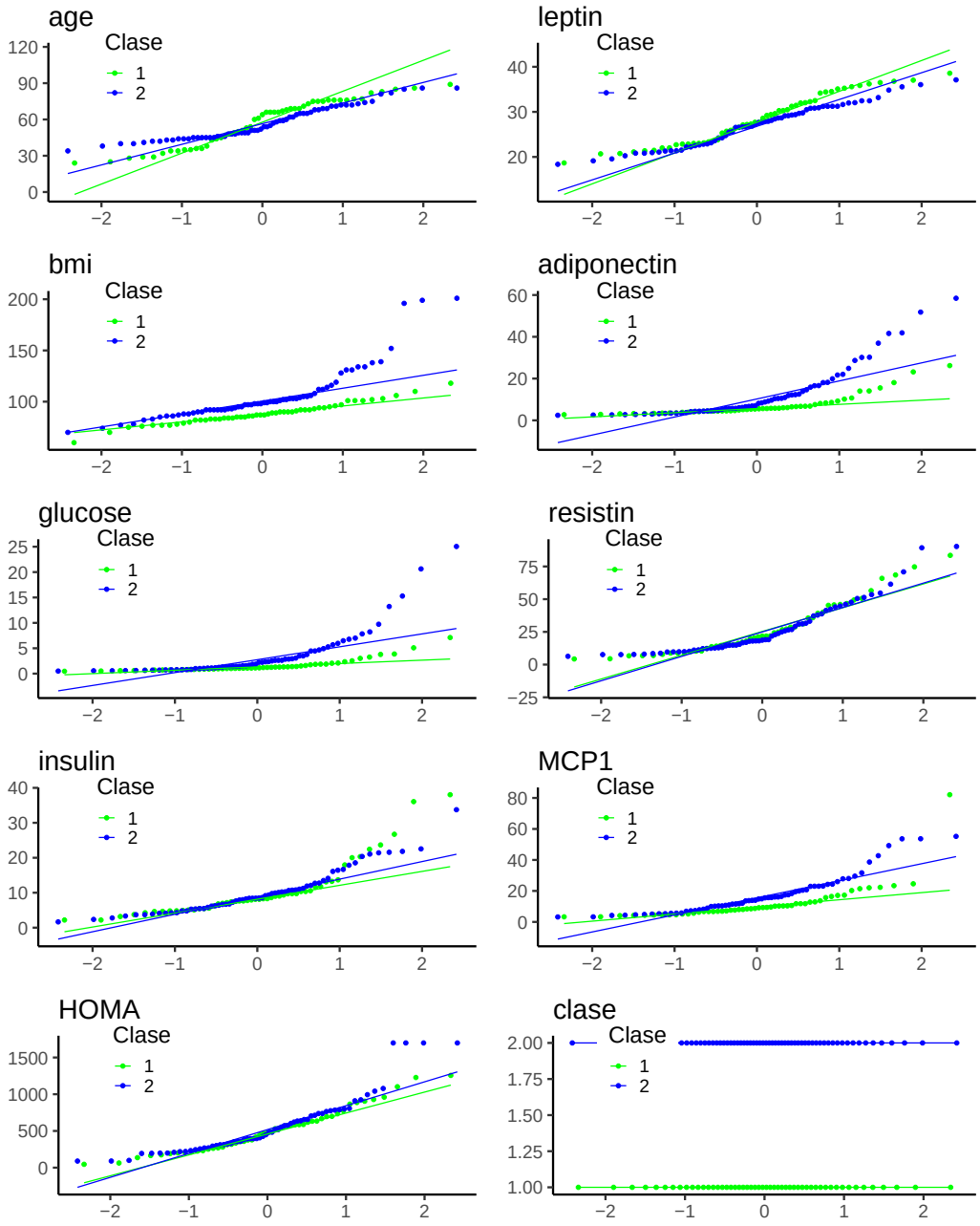

Fig. 11: QQplots R.

```
for (i in 1:10){
pdf (file = paste ('../images/qq', i, '.pdf', sep = ''), width = 6, height = 3)
print (ggplot (datos, aes (sample = datos[,i], colour = as.factor (clase))) +
                labs (x = NULL, y = NULL,
                      title = names (datos)[i], colour = 'Clase') +
                geom_qq () + geom_qq_line () + theme_classic (base_size = 20) +
                scale_colour_manual (values = c ('green', 'blue')) +
                theme   (legend.position = c (0.2, 0.85)))
dev.off ()
}
```

Listing 14: Código R generador de los QQplots.

```
for (i = 1 : 10)
        subplot (5, 2, i);
        qqplot (dataR2mat (dataR2mat (:, end) == 1, i));
        hold on;
        qqplot (dataR2mat (dataR2mat (:, end) == 2, i));
end
```

Listing 15: Código Matlab generador de los QQplots.

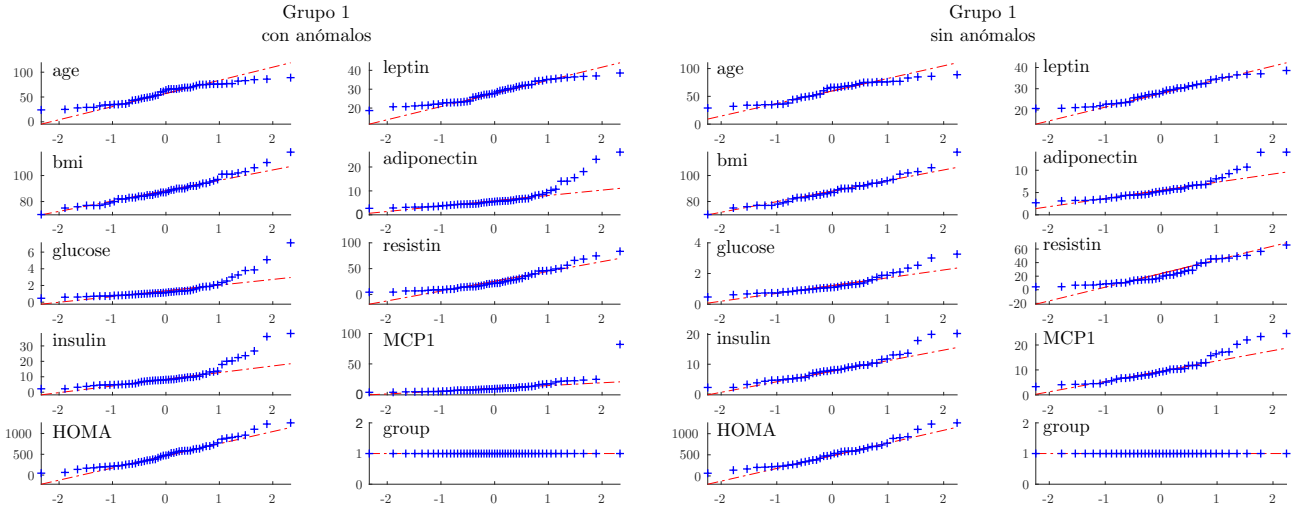

Fig. 12: QQplots Matlab Clase 1.

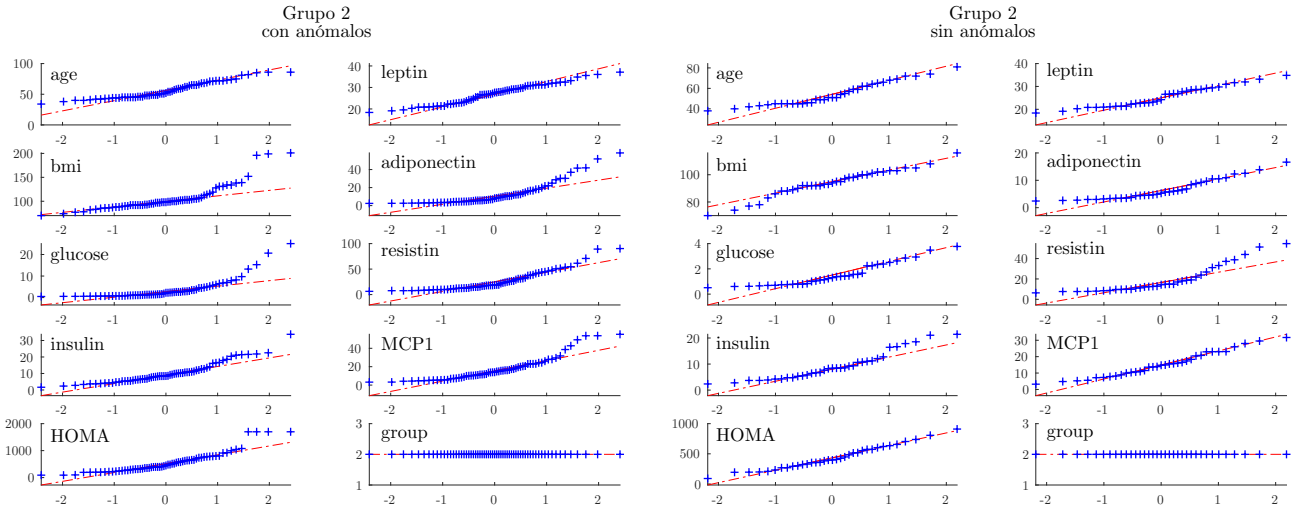

Fig. 13: QQplots Matlab Clase 2.

4.1.6 Corrplot

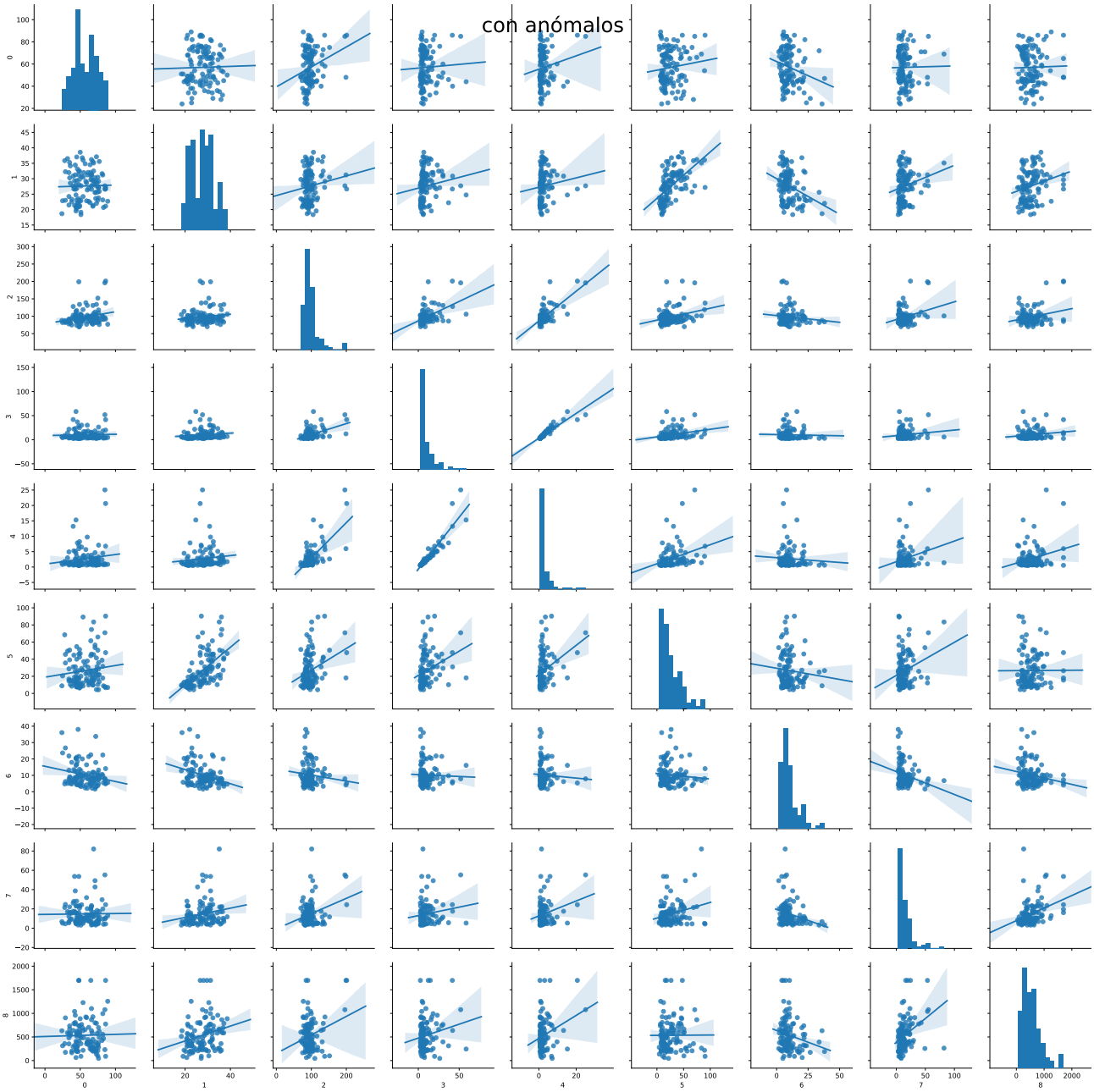

Fig. 14: Corrplot Python para datos con anomalias.

```
import pandas as pd
import seaborn as sns
dataframe = pd.DataFrame.from_records(x)
sns.pairplot (dataframe, kind = 'reg')
plt.suptitle ('con anómalos', fontsize = 30)
plt.savefig  ('../images/corrp.pdf', bbox_inches = 'tight', pad_inches = 0)
```

Listing 16: Código Python generador de los corrplots con datos anómalos.

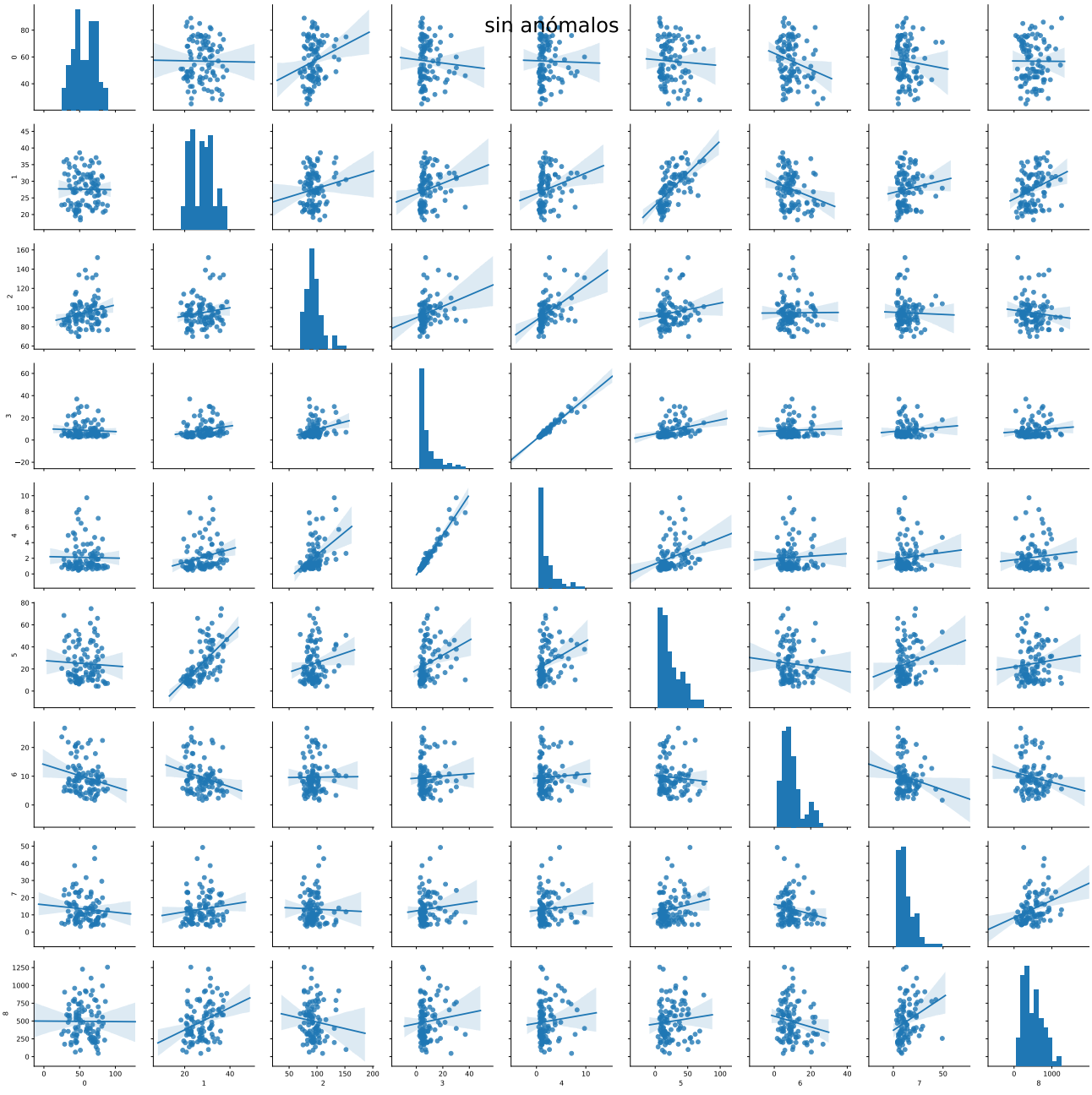

Fig. 15: Corrplot Python para datos sin anomalias.

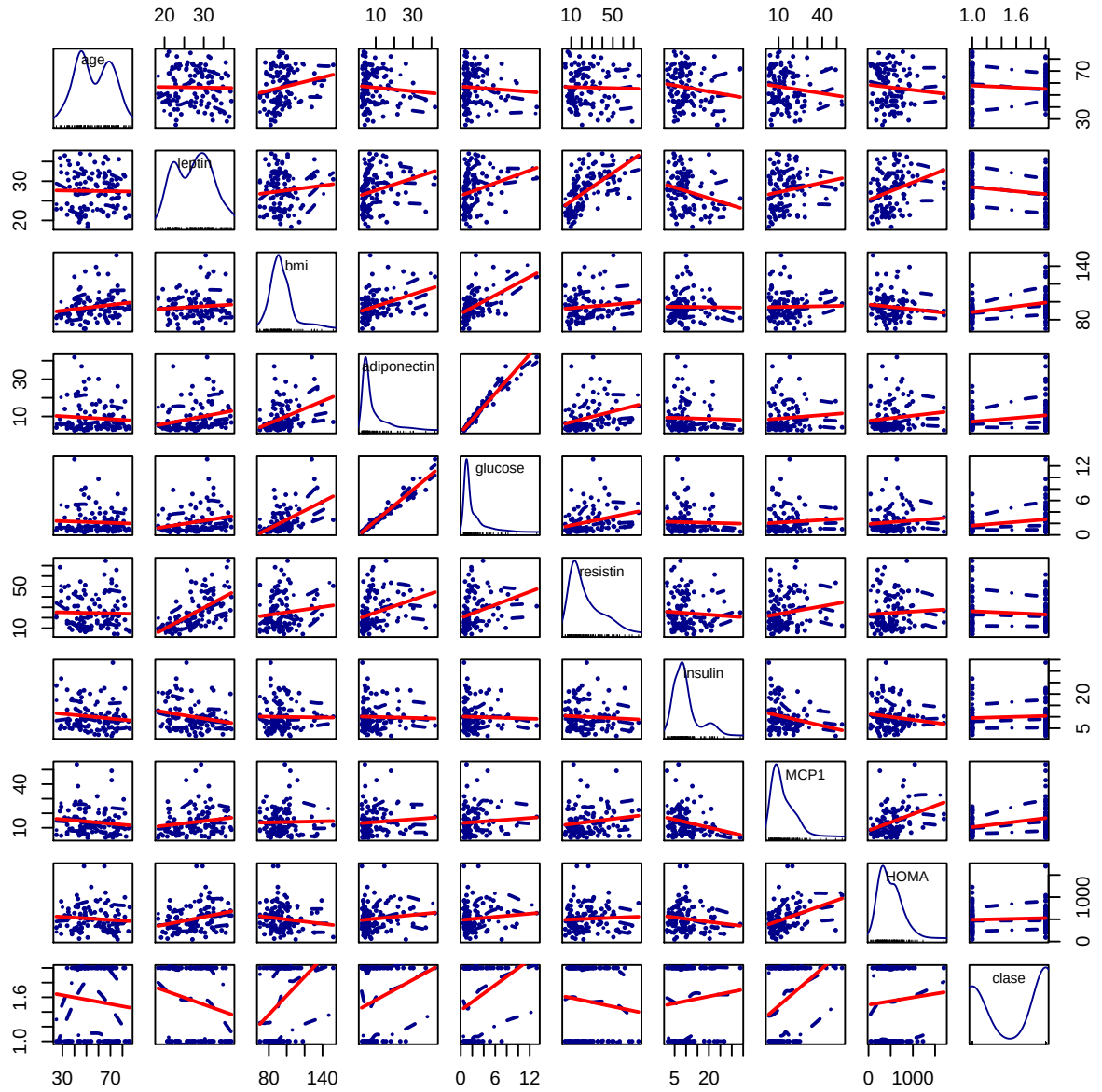

Fig. 16: Corrplot R para datos con anomalias.

```
1  library (car)
2  pdf ("../images/corrplot.pdf")
3  scatterplotMatrix (datos, regLine=list (col='red'), pch=20, cex=0.5, col='blue4')
4  dev.off ()
5
6  library (corrplot)
7  pdf ("../images/corrplot1.pdf")
8  M <- cor (na.omit (datos))
9  corrplot (M, method = 'number')
10 dev.off ()
```

Listing 17: Código R generador de los corrplots.

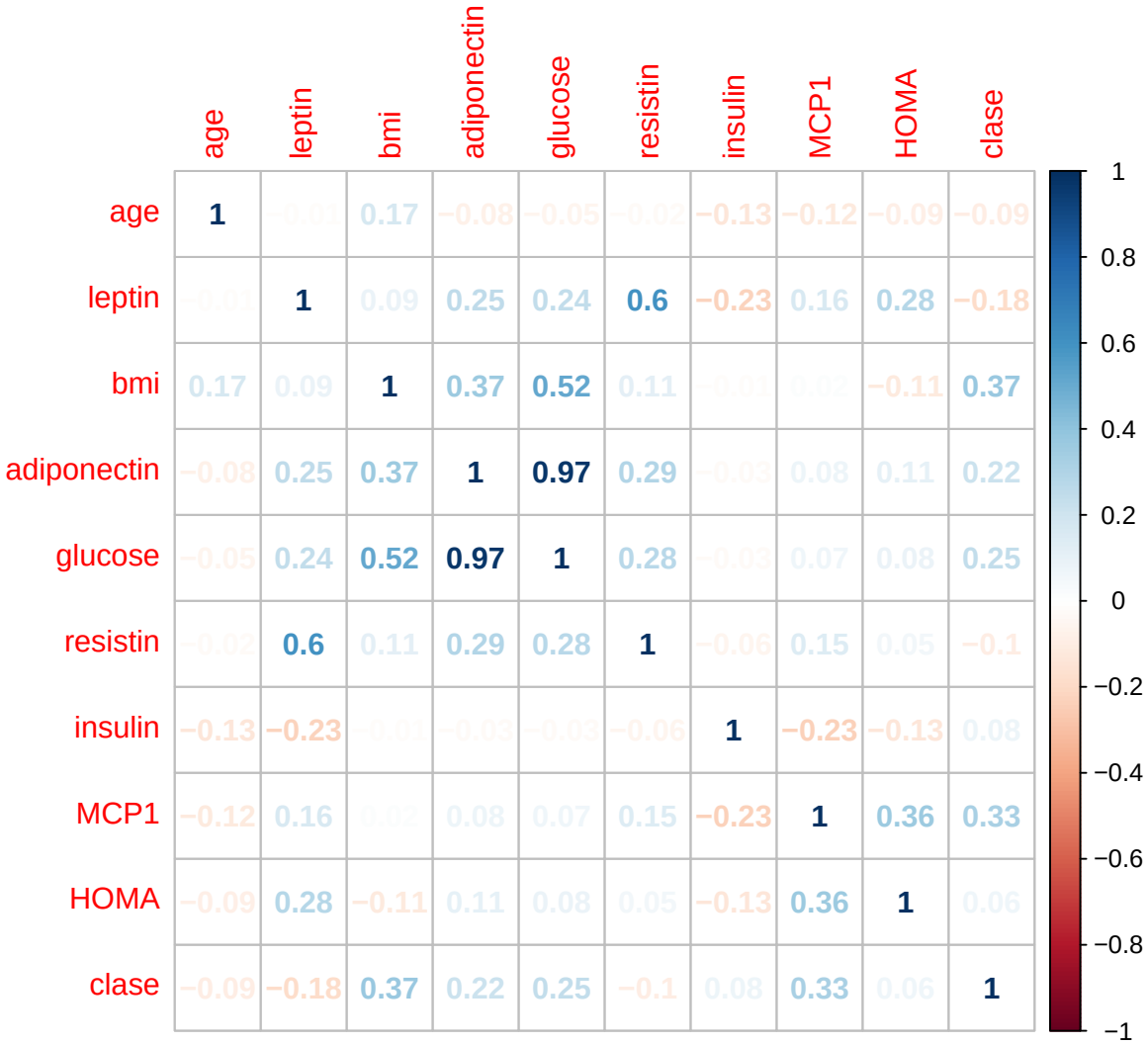

Fig. 17: Matriz de correlaciones en R.

```
corrplot (dataR2mat (:,1:end-1))
```

Listing 18: Código Matlab generador de los corrplots.

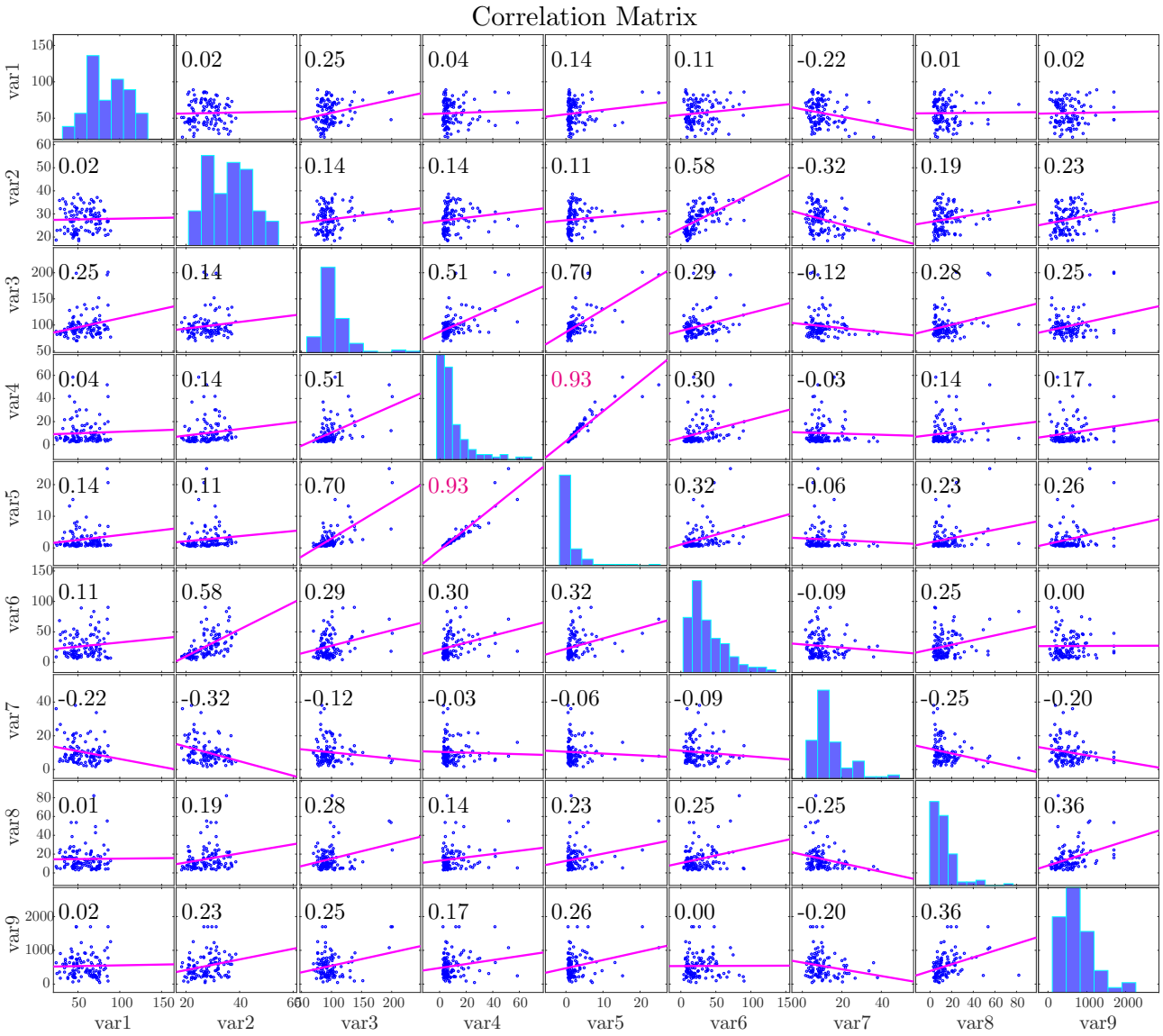

Fig. 18: Matriz de correlaciones en Matlab.

4.2 Extracción de características.

4.2.1 Filter Methods

```
#  Filter Methods
import sklearn.feature_selection as sk

Fscore, pval = sk.f_classif (x_no, y_no)
r1 = Fscore.argsort().argsort() #  fscore rank
print (r1+1)

import ReliefF as rl

r2 = rl.ReliefF (n_neighbors = 1) #  relieff rank
r2.fit(x_no, y_no)
r2 = r2.top_features
print (r2+1)

diferencias = abs (r1-r2)
media = np.mean (diferencias)
```

Listing 19: Métodos filter Python.

```
[4 5 9 6 7 3 1 8 2] -> fscore
[1 9 8 7 6 5 4 2 3] -> relieff
[3 4 1 1 1 2 3 6 1] -> diferencias
2.4444444444444446  -> media
```

Listing 20: Ranking de variables según los métodos filter Python.

```
[out   , rank]   = fscore  (x, y);
[RANKED, WEIGHT] = relieff (x, y, 1); % k = 1 neighbour
diferencia = abs(rank-RANKED')
media = mean (diferencia)
    var1  var2  var3  var4  var5  var6  var7  var8 var9
     3     5     4     8     2     9     1     6     7    -> fscore
     1     3     2     7     4     5     8     9     6    -> relieff
diferencias =
     2     2     2     1     2     4     7     3     1
media =
     2.6667
```

Listing 21: Ranking de variables Matlab.

```r
# Fscore
library (PredPsych)
rank (fscore (datos, 10, 1:9))
#  age  leptin  bmi adiponectin  glucose  resistin  insulin MCP1  HOMA
#    3       5    9           7        8         2        1    6     4

# Relieff
brary (CORElearn)
rank (attrEval (as.factor (clase)~., datos, 'Relief'))
#  age  leptin  bmi adiponectin  glucose  resistin  insulin  MCP1  HOMA
#    9       7    8           2        4         5        1     6     3

# Algunos de los posibles metodos
for (i in infoCore (what = "attrEval")){
    cat (i, '\r\t\t', unname (rank (attrEval (as.factor (clase)~., datos, i))),'\n')
}
# ReliefFequalK     9 3 8 4 6 5 2 7 1
# ReliefFexpRank    8 5 9 3 6 4 1 7 2
# ReliefFbestK      9 7 8 3 4 5 1 6 2
# Relief            9 7 8 2 4 5 1 6 3
# InfGain           7 4 9 5 8 2 1 6 3
# GainRatio         9 2 8 7 6 4.5 1 3 4.5
# MDL               7 4 9 5 8 3 1 6 2
# Gini              7 4 9 5 8 3 1 6 2
# MyopicReliefF     6 4 9 5 7 3 1 8 2
# Accuracy          6 4 9 5 7 3 1.5 8 1.5
# ReliefFmerit      8 3 9 5 6 4 1 7 2
# ReliefFdistance   8 4 9 5 6 3 1 7 2
# ReliefFsqrDistan  8 4 9 5 6 3 1 7 2
# DKM               7 3 9 6 8 2 1 5 4
# ReliefFexpC       8 5 9 3 6 4 1 7 2
# ReliefFavgC       8 5 9 3 6 4 1 7 2
# ReliefFpe         8 5 9 3 6 4 1 7 2
# ReliefFpa         8 5 9 3 6 4 1 7 2
# ReliefFsmp        8 5 9 3 6 4 1 7 2
# GainRatioCost     9 2 8 7 6 4.5 1 3 4.5
# DKMcost           7 4 9 5 8 3 2 6 1
#                .  .  .
```

Listing 22: Ranking de variables según distintos métodos en R.

4.2.2 Wrapper Methods

```
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.feature_selection import SequentialFeatureSelector

knn = KNeighborsClassifier (n_neighbors = 50)

sfs = SequentialFeatureSelector (knn,
                k_features = 4,
                forward = True,
                scoring = 'accuracy',
                cv = 10)

sfs.fit (x_no, y_no, custom_feature_names = labels)
print (sfs.k_score_)
print ('Sequential Forward  Selection', sfs.k_feature_names_, end = '\n\n')

sfs.forward = False

sfs.fit (x_no, y_no, custom_feature_names = labels)
print (sfs.k_score_)
print ('Sequential Backward Selection', sfs.k_feature_names_, end = '\n\n')
```

Listing 23: Aplicación métodos *wrapper* de selección características.

```
0.7054545454545454
Sequential Forward  Selection ('leptin', 'bmi', 'glucose', 'MCP1')

0.7094949494949495
Sequential Backward Selection ('leptin', 'bmi', 'glucose', 'insulin')
```

Listing 24: Resultados Python del filtrado mediante wrappers.

```r
# Sequential Feature Selector
library (mlr)
# Forward
sfs <- selectFeatures (
      learner    = makeLearner      ('classif.knn', k = 9, l = 3),
      task       = makeClassifTask  (data = datos, target = 'clase'),
      resampling = makeResampleDesc ("CV", iter = 50),
      control    = makeFeatSelControlSequential (method = "sfs", maxit = 100L))
# FeatSel result:
# Features (4): age, leptin, bmi, MCP1
# mmce.test.mean=0.1833333

# Backward
sbs <- selectFeatures (
      learner    = makeLearner      ('classif.knn', k = 9, l = 3),
      task       = makeClassifTask  (data = datos, target = 'clase'),
      resampling = makeResampleDesc ("CV", iter = 50),
      control    = makeFeatSelControlSequential (method = "sbs", maxit = 100L))
# FeatSel result:
# Features (4): age, leptin, bmi, MCP1
# mmce.test.mean=0.1800000
```

Listing 25: Resultados R del filtrado mediante wrappers.

```
fun = @(XT,yT,Xt,yt) ...
               (sum ((yt - classify (Xt,XT,yT,'quadratic'))~=0));
fsf = sequentialfs (fun, x, y,'direction', 'forward', 'cv', 10);
fsb = sequentialfs (fun, x, y,'direction', 'backward', 'cv', 10);
[fsf' fsb']'
ans =
   1   0   0   0   0   0   0   0   0   -> forward
   1   1   1   1   0   0   1   1   0   -> backward
```

Listing 26: Resultados Matlab del filtrado mediante wrappers.

4.2.3 PCA

```
from sklearn.preprocessing import StandardScaler
x_no = StandardScaler ().fit_transform (x_no) #  typify
from sklearn.decomposition import PCA
pca = PCA (n_components = 9)
principalComponents = pca.fit_transform(x_no)
evr = pca.explained_variance_ratio_
```

Listing 27: *Principal Component Analysis* Python.

```
[0.29146865 0.18490568 0.14125105 0.11727276 0.08486126 0.07999359
 0.06636991 0.03254865 0.00132847]
[0.29146865 0.47637432 0.61762537 0.73489813 0.81975939 0.89975298
0.96612289 0.99867153 1.        ]
```

Listing 28: Varianza explicada por componente y suma acumulada Python.

```
pca <- prcomp (datos[,1:9], center = T, scale. = T, rank. = 9)
summary (pca)
```

Listing 29: *Principal Component Analysis* R.

```
Importance of components:
                PC1    PC2    PC3    PC4    PC5    PC6    PC7     PC8     PC9
Std deviation   1.7475 1.2393 1.082 1.048 0.8528 0.8144 0.66261 0.53101 0.17555
Propor. of Var. 0.3393 0.1707 0.130 0.122 0.0808 0.0737 0.04878 0.03133 0.00342
Cum. Var.       0.3393 0.5100 0.640 0.762 0.8428 0.9165 0.96525 0.99658 1.00000
```

Listing 30: Varianza explicada por componente y suma acumulada R.

```
x = zscore (x);
[coeff, score, latent] = pca (x);
pareto (latent)
```

Listing 31: *Principal Component Analysis* Matlab y pareto.

4.2.4 Pareto

```
ax.bar (range (len (evr)), evr)
ax.set_ylim (top=1)
ax1 = ax.twinx ()
ax1.set_ylim (top=100)
ax1.plot (range (len (evr)), np.cumsum (evr)*100, marker = '.', color = 'red')
fig.suptitle ('Pareto Python', fontsize = 16)
fig.savefig  ('../images/pareto.pdf', bbox_inches = 'tight', pad_inches = 0)
```

Listing 32: Código generador del diagrama de Pareto en Python.

```
pdf ("../images/pareto.pdf", width = 7, height = 5.5)
x  <- pca[['sdev']]^2
cx <- cumsum (x)
par (mar = c(3,3,4,3))
pc <- barplot (x, names.arg = dimnames (pca[['rotation']])[[2]],
               border = NA, axes = F, main = 'Pareto R',
               ylim = c(0, 1.05*max(cx, na.rm = T)), col = 'blue4'
)
lines (pc, cx, type = 'b', pch = 19, col="red")
box  (col    = 'black')
axis (side   = 2,
      at     = c (0, round (x[c (1,2,4,6,8,9)], 1)),
      las    = 2, cex.axis = 0.8,
)
axis (side   = 4,
      at     = c(0, cx[1:8]),
      labels = paste (c (0, round (cx[1:8]/max (cx) * 100)) ,"%",sep=""),
      las    = 2, cex.axis = 0.8
)
dev.off ()
```

Listing 33: Código generador del diagrama de Pareto en R.

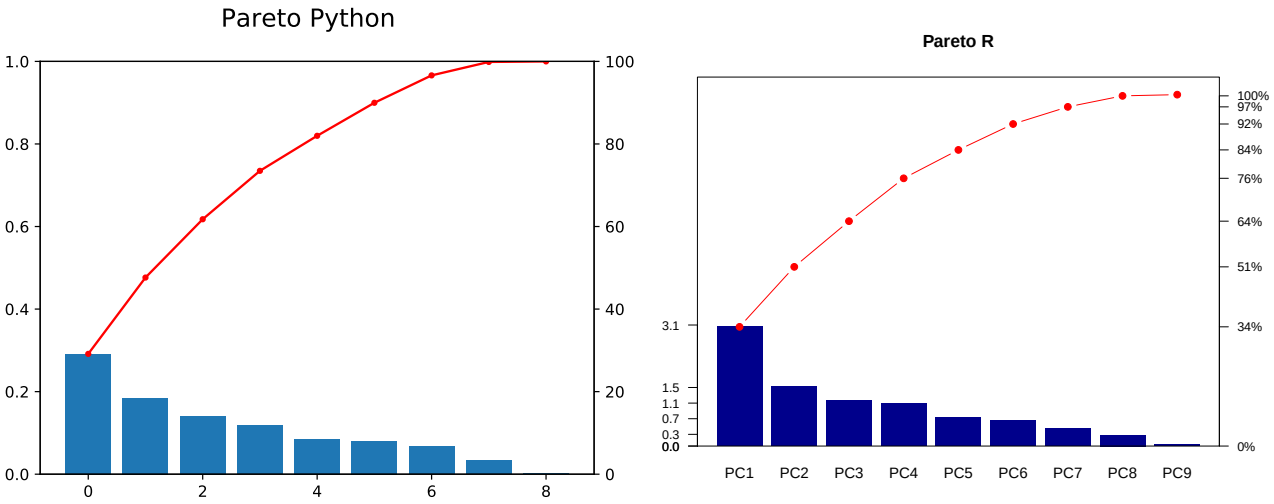

Fig. 19: Diagrama de Pareto en Python y R.

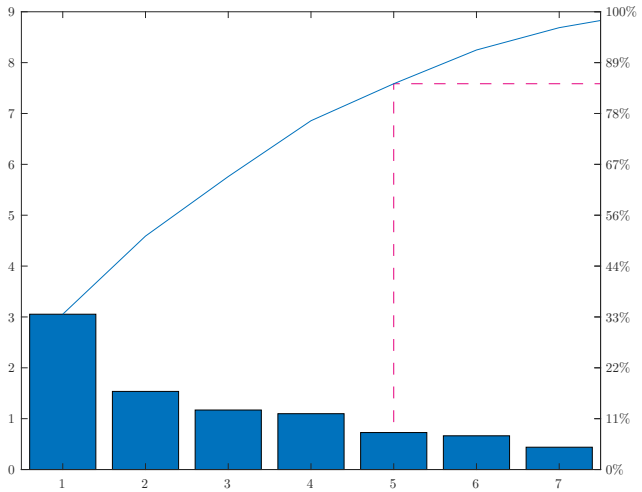

Fig. 20: Diagrama de Pareto en Matlab.

4.2.5 Biplot

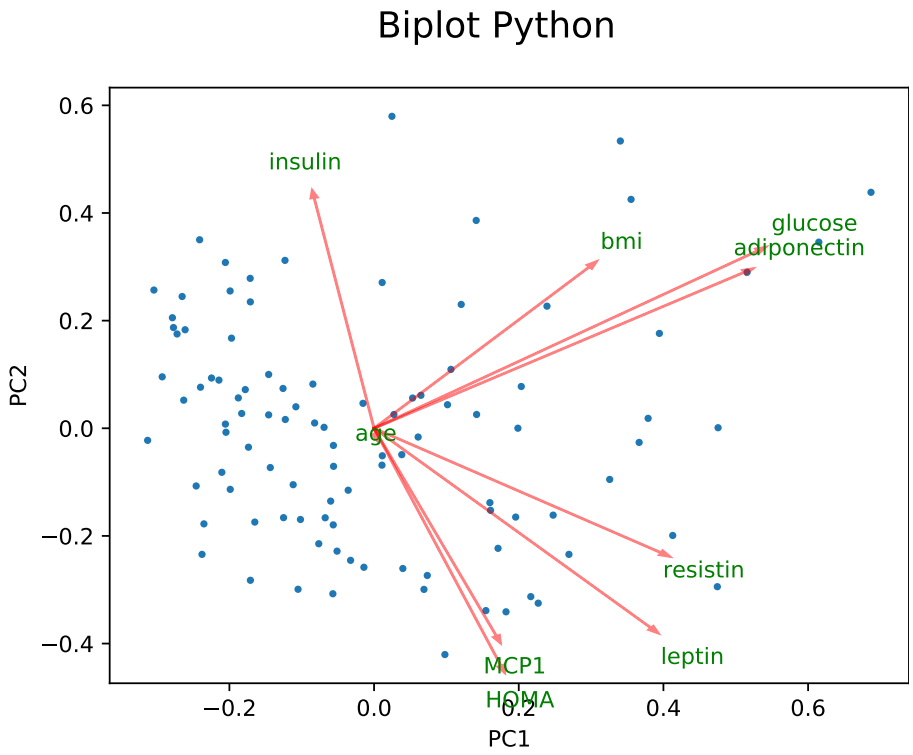

Fig. 21: Biplot Python.

```
1  def biplot (score,coeff,labels=None):
2      """from https://ostwalprasad.github.io/machine-learning/PCA-using-python.html"""
3      xs = score[:,0]; ys = score[:,1]
4      n = coeff.shape[0]
5      scalex = 1.0/(xs.max() - xs.min())
6      scaley = 1.0/(ys.max() - ys.min())
7      plt.scatter(xs * scalex,ys * scaley,s=5)
8      plt.suptitle ('Biplot Python', fontsize = 16)
9      for i in range(n):
10         plt.arrow(0, 0, coeff[i,0], coeff[i,1],color='r',alpha=0.5, head_width=0.01)
11         if labels is None:
12             plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = '↘
               green', ha = 'center', va = 'center')
13         else:
14             plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, labels[i], color = 'g', ha↘
                = 'center', va = 'center')
15     plt.xlabel("PC{}".format(1)), plt.ylabel("PC{}".format(2))
16     plt.savefig  ('../images/biplotpca.pdf', bbox_inches = 'tight', pad_inches = 0)
17 biplot (principalComponents[:,0:2], np.transpose(pca.components_[0:2, :]), labels)
```

Listing 34: Código generador del Biplot en Python.

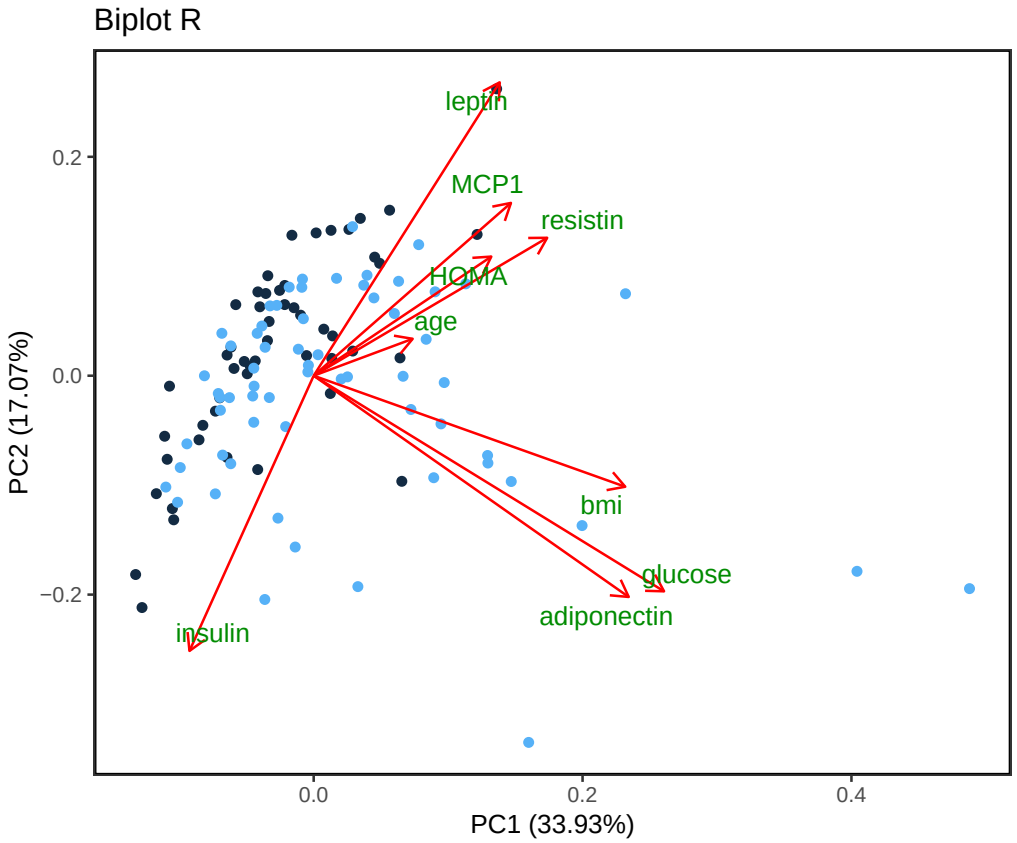

Fig. 22: Biplot R.

```
library (ggfortify)
pdf ("../images/biplot.pdf", width = 6, height = 5)
autoplot (pca, data = datos, colour = 'clase',
          loadings  = T,
          main      = 'Biplot R',
          loadings.label        = T,
          loadings.label.repel  = T,
          loadings.label.colour = 'green4',
) +
theme_bw () +
theme (panel.grid.major = element_blank(),
       panel.grid.minor = element_blank(),
       panel.background = element_rect(colour = "black", size = 1),
       legend.position  = 'none'
)
```

Listing 35: Código generador del Biplot en R.

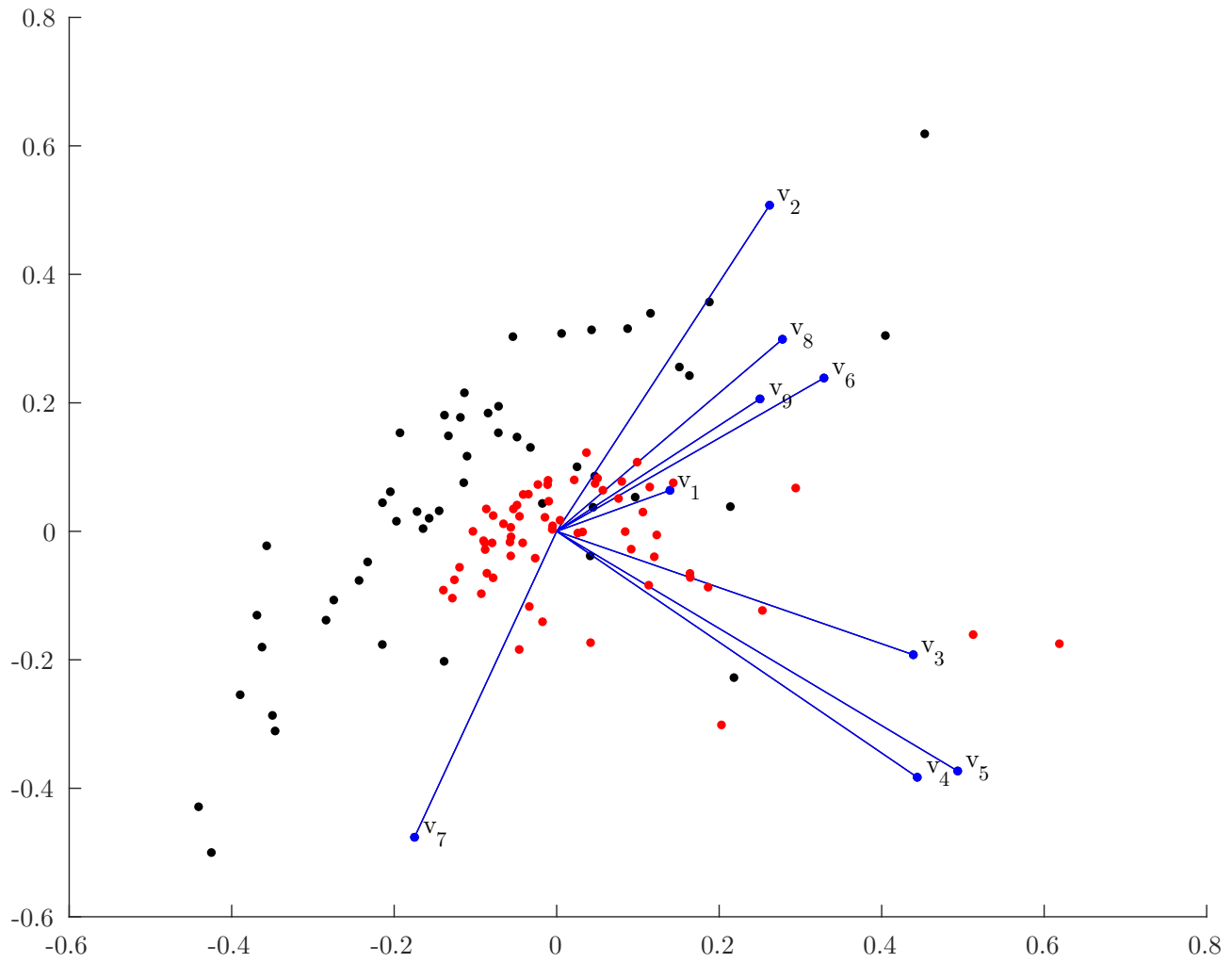

Fig. 23: Biplot Matlab.

```
hold on
biplot (coeff (:,1:2), 'score', score(1:51,1:2), ...
        'Color', 'k', 'MarkerSize',10)
biplot (coeff (:,1:2), 'score', score(51:end,1:2), ...
        'varlabels', varnames, 'MarkerSize',10)
hold off
```

Listing 36: Código generador del Biplot en Matlab.

5 Modelos de Clasificación

5.1 Clasificación Lineal

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA ()
score = cross_val_score (lda, x, y, cv = 10)
print ('Linear puntuación CV media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))

score = cross_val_score (lda, x, y, cv = KFold (n_splits = 10, shuffle = True))
print ('Linear puntuación KF media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))

score = cross_val_score (lda, x, y, cv = ShuffleSplit (n_splits = 10))
print ('Linear puntuación SS media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))

score = cross_val_score (lda, x, y, cv = LeaveOneOut ())
print ('Linear puntuación LO media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))
```

Listing 37: Python validación del modelo lineal.

```
Linear puntuacion CV media: 0.75 std: 0.13
Linear puntuacion KF media: 0.75 std: 0.10
Linear puntuacion SS media: 0.71 std: 0.14
Linear puntuacion LO media: 0.76 std: 0.43
```

Listing 38: Python validación según distintos métodos de partición.

```r
# Linear Discriminant Analysis
it <- 1000
ldascores <- rep (NA, times = it)
p <- 0.7 # partition
cat ('LDA\n')
pb <- txtProgressBar (min = 0, max = it, initial = 0, char = '|', style = 3)
for (i in 1:it){
train.samples <- createDataPartition (datos$clase, p = p, list = F)

train.data    <- datos[ train.samples,]
test.data     <- datos[-train.samples,]

preproc.param <- preProcess (train.data, method = c ("center", "scale"))

train.trans   <- predict (preproc.param, train.data)
test.trans    <- predict (preproc.param, test.data)

mdl <- lda (clase~., data = train.trans)

prd <- predict (mdl, test.trans)

ldascores[i]  <- mean (prd$class == test.trans$clase)
setTxtProgressBar (pb, i)
}
```

Listing 39: R análisis lineal discriminante.

```
           vars    n  mean   sd median trimmed  mad  min  max range  skew
ldascores     1 1000 0.72 0.08   0.73    0.72 0.10 0.47 0.97  0.50 -0.09
```

Listing 40: R puntuación de mil evaluaciones.

```
1
2  n    = length (x);
3  p    = 0.3;
4  cvp = cvpartition (n, 'HoldOut', p);
5  iteraciones = 10000;
6  resultados  = zeros (iteraciones,1);
7  for i = 1 : iteraciones
8          cvp = cvpartition (n, 'HoldOut', p);
9          Mdl = fitcdiscr (x (cvp.training), y (cvp.training), ...
10                 'DiscrimType', 'linear');
11         [label, score, cost] = predict (Mdl, x (cvp.test));
12         dif = abs (label - y (cvp.test));
13         errores  = sum (dif == 1); aciertos = sum (dif == 0);
14         resultados(i) = (aciertos/(aciertos+errores)) * 100;
15 end
16 [muHat,sigmaHat,muCI,sigmaCI] = normfit (resultados,.01)
17
18 % % Linear
19 % muHat =
20 %         52.586
21 % sigmaHat =
22 %         7.5288
23 % muCI =
24 %         52.392
25 %          52.78
26 % sigmaCI =
27 %          7.394
28 %         7.6683
```

Listing 41: Matlab análisis lineal discriminante.

5.2 Clasificación Cuadrática

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA

qda = QDA ()
score = cross_val_score (qda, x, y, cv = 10)
print ('Quadratic puntuación CV media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))

score = cross_val_score (qda, x, y, cv = KFold (n_splits = 10, shuffle = True))
print ('Quadratic puntuación KF media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))

score = cross_val_score (qda, x, y, cv = ShuffleSplit (n_splits = 10))
print ('Quadratic puntuación SS media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))

score = cross_val_score (qda, x, y, cv = LeaveOneOut ())
print ('Quadratic puntuación LO media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))
```

Listing 42: Python validación del modelo cuadrático.

```
Quadratic puntuacion CV media: 0.66 std: 0.19
Quadratic puntuacion KF media: 0.76 std: 0.09
Quadratic puntuacion SS media: 0.76 std: 0.14
Quadratic puntuacion LO media: 0.73 std: 0.44
```

Listing 43: Python validación según distintos métodos de partición.

```r
qdascores <- rep (NA, times = it)
cat ('\nQDA\n')
pb <- txtProgressBar (min = 0, max = it, initial = 0, char = '|', style = 3)
for (i in 1:it){
train.samples <- createDataPartition (datos$clase, p = p, list = F)

train.data    <- datos[ train.samples,]
test.data     <- datos[-train.samples,]

preproc.param <- preProcess (train.data, method = c ("center", "scale"))

train.trans   <- predict (preproc.param, train.data)
test.trans    <- predict (preproc.param, test.data)

mdl <- qda (clase~., data = train.trans)

prd <- predict (mdl, test.trans)

qdascores[i]  <- mean (prd$class == test.trans$clase)

setTxtProgressBar (pb, i)
}
```

Listing 44: R análisis cuadrático discriminante.

```
     vars         n mean   sd median trimmed  mad  min  max range  skew
qdascores    2 1000 0.69 0.07   0.70    0.69 0.10 0.43 0.87  0.43 -0.17
```

Listing 45: R puntuación de mil evaluaciones.

```
1
2  n   = length (x);
3  p   = 0.3;
4  cvp = cvpartition (n, 'HoldOut', p);
5  iteraciones = 10000;
6  resultados  = zeros (iteraciones,1);
7  for i = 1 : iteraciones
8          cvp = cvpartition (n, 'HoldOut', p);
9          Mdl = fitcdiscr (x (cvp.training), y (cvp.training), ...
10                 'DiscrimType', 'quadratic');
11         [label, score, cost] = predict (Mdl, x (cvp.test));
12         dif = abs (label - y (cvp.test));
13         errores  = sum (dif == 1); aciertos = sum (dif == 0);
14         resultados(i) = (aciertos/(aciertos+errores)) * 100;
15 end
16 [muHat,sigmaHat,muCI,sigmaCI] = normfit (resultados,.01)
17
18 % % Quadratic
19 % muHat =
20 %         62.252
21 % sigmaHat =
22 %         8.3548
23 % muCI =
24 %         62.037 62.468
25 % sigmaCI =
26 %         8.2051 8.5096
```

Listing 46: Matlab análisis lineal discriminante.

5.3 Clasificación KNN

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier (n_neighbors = 9)
score = cross_val_score (knn, x, y, cv = 10)
print ('KNN puntuación CV media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))

score = cross_val_score (knn, x, y, cv = KFold (n_splits = 10, shuffle = True))
print ('KNN puntuación KF media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))

score = cross_val_score (knn, x, y, cv = ShuffleSplit (n_splits = 10))
print ('KNN puntuación SS media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))

score = cross_val_score (knn, x, y, cv = LeaveOneOut ())
print ('KNN puntuación LO media: %.2f std: %.2f'
        %(np.mean (score), np.std (score)))
```

Listing 47: Python validación del modelo KNN.

```
KNN puntuacion CV media: 0.47 std: 0.12
KNN puntuacion KF media: 0.47 std: 0.15
KNN puntuacion SS media: 0.47 std: 0.13
KNN puntuacion LO media: 0.43 std: 0.50
```

Listing 48: Python validación según distintos métodos de partición.

```r
knnscores <- rep (NA, times = it)
library (class)
cat ('\nKNN\n')
pb <- txtProgressBar (min = 0, max = it, initial = 0, char = '|', style = 3)
for (i in 1:it){
train.samples <- createDataPartition (datos$clase, p = p, list = F)

train.data    <- datos[ train.samples,]
test.data     <- datos[-train.samples,]

preproc.param <- preProcess (train.data, method = c ("center", "scale"))

train.trans   <- predict (preproc.param, train.data)
test.trans    <- predict (preproc.param, test.data)

prd <- knn (train = train.trans[1:9],
            cl    = train.trans$clase,
            test  = test.trans[1:9],
            k     = 1)

knnscores[i]  <- mean (prd == test.trans$clase)

setTxtProgressBar (pb, i)
}
```

Listing 49: R *K nearest neighbours.*

```
     vars         n mean   sd median trimmed  mad  min  max range  skew
knnscores    3 1000 0.66 0.07   0.67    0.66 0.05 0.43 0.90  0.47 -0.07
```

Listing 50: R puntuación de mil evaluaciones.

```
n    = length (x);
p    = 0.3;
cvp = cvpartition (n, 'HoldOut', p);
iteraciones = 10000;
resultados  = zeros (iteraciones,1);
for i = 1 : iteraciones
        cvp = cvpartition (n, 'HoldOut', p);
        Mdl = fitcknn (x (cvp.training), y (cvp.training), ...
                'NumNeighbors', 1);
        [label, score, cost] = predict (Mdl, x (cvp.test));
        dif = abs (label - y (cvp.test));
        errores  = sum (dif == 1); aciertos = sum (dif == 0);
        resultados(i) = (aciertos/(aciertos+errores)) * 100;
end
[muHat,sigmaHat,muCI,sigmaCI] = normfit (resultados,.01)

% k = 1
% muHat =
%        53.105
% sigmaHat =
%        7.1829
% muCI =
%         52.92
%         53.29
% sigmaCI =
%        7.0543
%         7.316
```

Listing 51: Matlab KNN.

6 Postámbulo

6.1 Comparación *LDA*, *QDA*, *KNN* R.

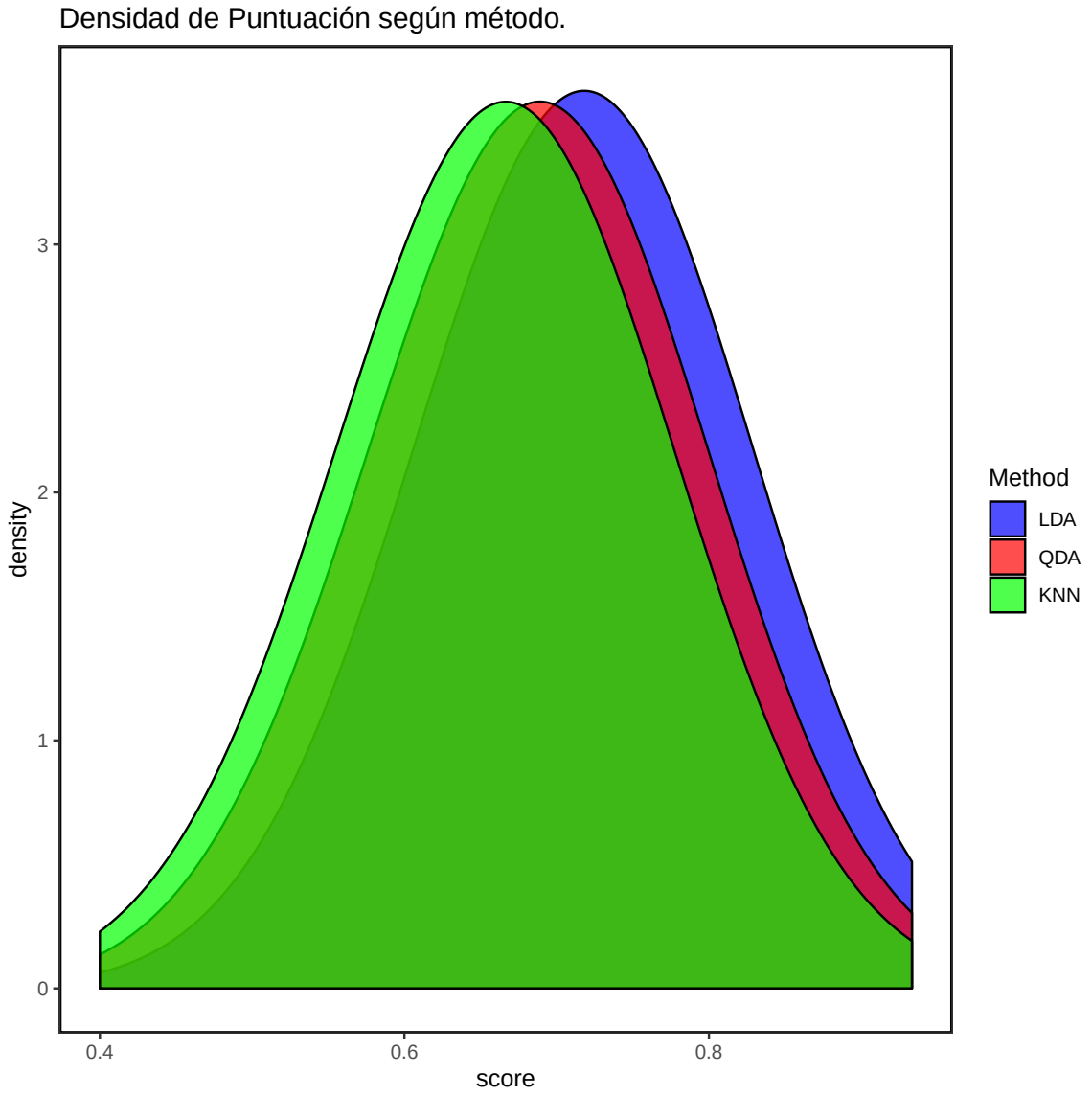

Fig. 24: Comparación de mil evaluaciones de cada uno de los métodos de clasificación en R.

6.2 Puntuación *knn* vs. número de vecinos.

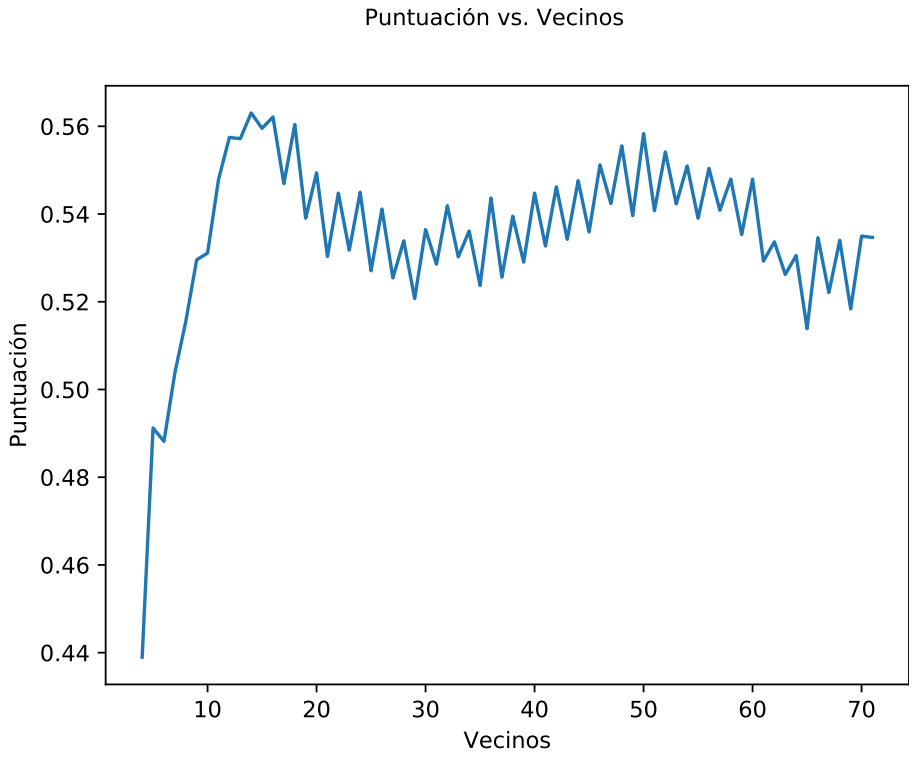

Fig. 25: Rendimineto decreciente según aumenta el número de vecinos.

```
score = [None]*(vecinos)
for i in range (2,vecinos):
    print ('n_neighbors = %i'% (i), end = '\r')
    iteraciones = 1000
    error =  [None]*iteraciones
    for j in range (0, iteraciones):
        X_train, X_test, y_train, y_test = train_test_split (x, y, test_size = 0.3)
        knn = KNeighborsClassifier (n_neighbors = i, n_jobs = -1)
        knn.fit (X_train, y_train)
        error[j] = np.sum (abs (knn.predict (X_test) - y_test))/ len (y_test)
    score[i] = np.mean (error)

plt.plot (range (2, vecinos+2), score)

plt.suptitle ('Puntuación vs. Vecinos', fontsize = 10)
plt.suptitle ('puntuación vs. vecinos', fontsize = 10)
plt.xlabel ('vecinos')
plt.ylabel ('puntuación')
plt.show ()
```

Listing 52: Evolución de puntuación según número de vecinos.

6.3 *Benchmarking*

```
benchmark (
'1 load'= {
        datos <- read.table ('../../data.csv', sep = ',', header = T)
        datos <- na.omit (datos)
        datos <- filter_all (datos, all_vars (. <= quantile (., 0.99, na.rm = T)))
        p <- 0.7
},
'2 part'= {
        train.samples <- createDataPartition (datos$clase, p = p, list = F)
        train.data    <- datos[ train.samples,]
        test.data     <- datos[-train.samples,]
        preproc.param <- preProcess (train.data, method = c ("center", "scale"))
        train.trans   <- predict (preproc.param, train.data)
        test.trans    <- predict (preproc.param, test.data)
},
'3 lda' = {
        mdl <- lda (clase~., data = train.trans)
        prd <- predict (mdl, test.trans)
        mean (prd$class == test.trans$clase)
},
'4 qda' = {
        mdl <- qda (clase~., data = train.trans)
        prd <- predict (mdl, test.trans)
        mean (prd$class == test.trans$clase)
},
'5 knn' = {
        prd <- knn (train = train.trans[1:9],
                    cl    = train.trans$clase,
                    test  = test.trans[1:9],
                    k     = 1
        )
        mean (prd == test.trans$clase)
},
replications = 1000,
columns = c ("test", "replications", "elapsed",
             "relative", "user.self", "sys.self")
)
```

Listing 53: Código R para evaluar el tiempo de ejecucción.

```
import pytest

# Absolute times
@measure
def loa ():
    global x_no, y_no
    from preprocessing import x_no, y_no

@measure
def par ():
    global X_train, X_test, y_train, y_test
    X_train, X_test, y_train, y_test = train_test_split (x_no, y_no)

@measure
def lda ():
    lda = LDA ()
    lda.fit (X_train, y_train)
    error = np.sum (abs (lda.predict (X_test) - y_test)) / len (y_test)

@measure
def qda ():
    qda = QDA ()
    qda.fit (X_train, y_train)
    error = np.sum (abs (qda.predict (X_test) - y_test)) / len (y_test)

@measure
def knn ():
    knn = KNeighborsClassifier (n_neighbors = 1)
    knn.fit (X_train, y_train)
    error = np.sum (abs (knn.predict (X_test) - y_test)) / len (y_test)

loa (), par (), lda (), qda (), lda ()
compare (lda, qda, 1000)
compare (qda, knn, 1000)
compare (lda, knn, 1000)
```

Listing 54: Código Python para evaluar el tiempo de ejecucción.

```
    test replications elapsed relative user.self sys.self
1   1 load         1000   4.803    8.486     4.712    0.090
2   2 part         1000  13.267   23.440    13.201    0.061
3   3 lda          1000   3.359    5.935     3.346    0.013
4   4 qda          1000   3.480    6.148     3.455    0.025
5   5 knn          1000   0.566    1.000     0.562    0.004
```

Listing 55: Tiempo de ejecución en R.

```
'loa' ((), {}) 0.00376701354980468750 sec
'par' ((), {}) 0.00057792663574218750 sec
'lda' ((), {}) 0.00213122367858886719 sec
'qda' ((), {}) 0.00131011009216308594 sec
'lda' ((), {}) 0.00145196914672851562 sec
Result: qda is 1.516303298299778 times faster than lda
Result: qda is 2.87883278273268 times faster than knn
Result: lda is 1.918905609112522 times faster than knn
```

Listing 56: Tiempo de ejecución en Python.

```
OS: macOS Sierra 10.12.6 16G1618 x86_64
Host: iMac12,2
Kernel: 16.7.0
Uptime: 4d 23h
Packages: 292 (port), 233 (brew)
Shell: zsh 5.2
Resolution: 2560x1440
DE: Aqua
WM: Kwm
Terminal: kitty
CPU: Intel i5-2500S (4) @ 2.70GHz
GPU: AMD Radeon HD 6770M
Memory: 7027MiB / 12288MiB
Disk (/): 806G / 931G (87%)
```

Listing 57: Características de la máquina usada.

```
tic
load('Data.mat');
dataR2mat = dataR2mat(~any(ismissing(dataR2),2),:);
dataR2mat=dataR2mat(~any(isoutlier(dataR2mat(:,1:end-1)),2),:);
datos = dataR2mat(:,1:end-1);
clasificacion = dataR2mat(:,end);
toc
loa = toc;

tic
n = length(datos);
p = 0.3;
cvp = cvpartition(n,'HoldOut',p) ;
tr_datos = datos(cvp.training);
te_datos = datos(cvp.test);
tr_class = clasificacion(cvp.training);
te_class = clasificacion(cvp.test);
toc
par = toc;

tic
Mdl = fitcdiscr(tr_datos,tr_class, ...
    'DiscrimType','linear');
tr_predict = predict(Mdl,tr_datos);
te_predict = predict(Mdl,te_datos);
toc
lda = toc;

tic
Mdq = fitcdiscr(tr_datos,tr_class, ...
    'DiscrimType','quadratic');
tr_predict = predict(Mdq,tr_datos);
te_predict = predict(Mdq,te_datos);
toc
qda = toc;

```

```
37 tic
38 k = 1; %1, 45, r = sqrt(n)
39 Mdk = fitcknn(tr_datos,tr_class, ...
40     'NumNeighbors',k);
41 tr_predict = predict(Mdk,tr_datos);
42 te_predict = predict(Mdk,te_datos);
43 toc
44 knn = toc;
```

Listing 58: Código Matlab para medir los tiempos.

6.4 Curva ROC

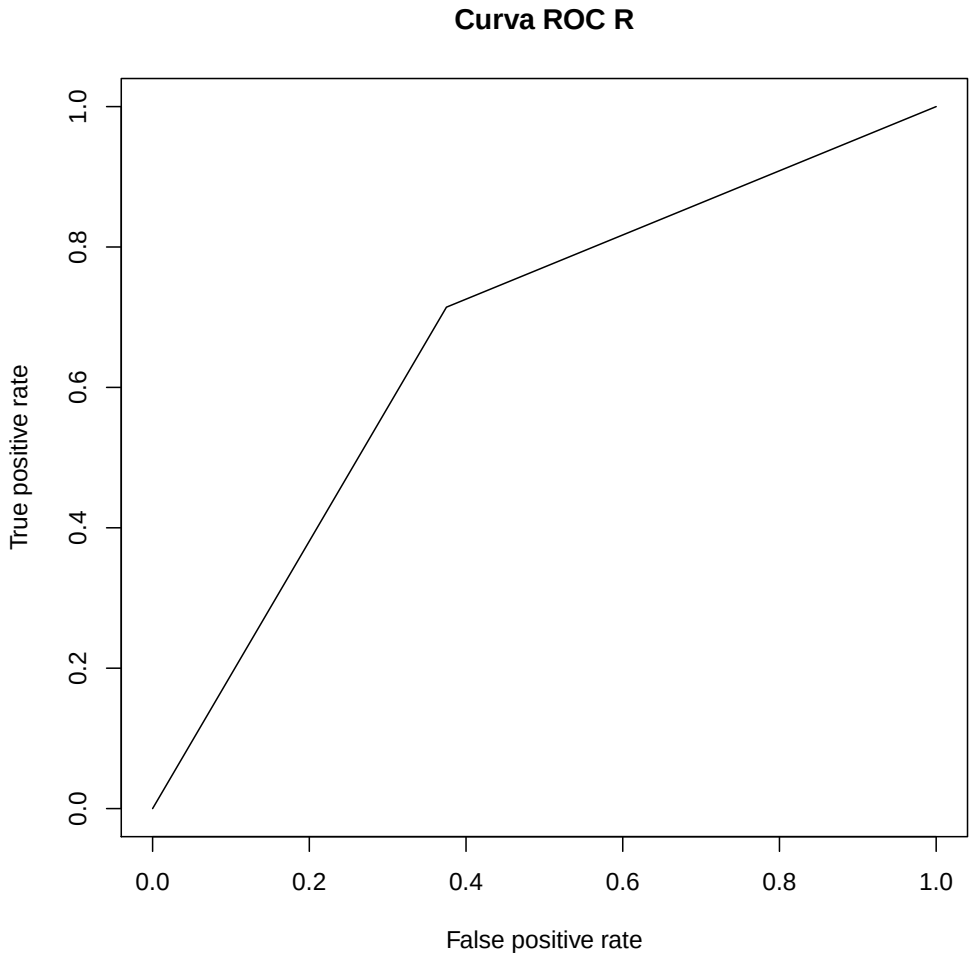

Fig. 26: ROC R.

```
1  suppressPackageStartupMessages (library (ROCR))
2  pred <- prediction (as.numeric (prd$class), as.numeric (test.trans$clase))
3  rocs <- performance (pred, "tpr", "fpr")
4
5  pdf ("../images/roc.pdf")
6  plot(rocs)
7  title ('Curva ROC R')
8  dev.off ()
9
10 auc_ROCR <- performance(pred, measure = "auc")
11 auc_ROCR@y.values[[1]]
12 suppressPackageStartupMessages (library (SDMTools))
13 mat <- confusion.matrix(as.numeric (prd$class) - 1 , as.numeric (test.trans$clase) -↘
       1,threshold=0.5)
```

Listing 59: Código R para generar la curva ROC y la AUC.

```
for i=1:p
k = 1; %1, 45, r = sqrt(n)
Mdk = fitcknn(datos(cvp.training(i)),clasificacion(cvp.training(i)), ...
    'NumNeighbors',k);
[labelK,score,cost] = predict(Mdk,datos(cvp.test(i)));
dif = abs(labelK-clasificacion(cvp.test(i)));
errores = sum(dif==1); % si la diferencia es de 1 nos hemos equivocado
aciertos = sum(dif==0); % si la diferencia es de 0 hemos acertado
PaciertosK = (aciertos/(aciertos+errores))*100;
Paciertos=[Paciertos PaciertosK];

[FP,VP] = perfcurve(labelK, clasificacion(cvp.test(i)), '1');
FP_matrix = [FP_matrix FP];
VP_matrix = [VP_matrix VP];
end
FP_media = mean(FP_matrix,2);
VP_media = mean(VP_matrix,2);
[media_K, desviacion_K] = normfit(Paciertos);

% METRICAS DE ERROR

% Curva ROC
figure;
plot(FP_media,VP_media), xlabel('Ratio Falso Positivo'), ylabel('Ratio
    Verdadero Positivo'), title('Curva ROC')

% Hacemos la matriz de confusion
C_K=confusionmat(clasificacion(cvp.test(1)),labelK);
%figure(2);confusionchart(C_K);title('Matriz de confusion KFold');
%cp_K = classperf(labelL,clasificacion(cvp.test(1)));
```

Listing 60: Código Matlab para generar la curva ROC y la AUC.

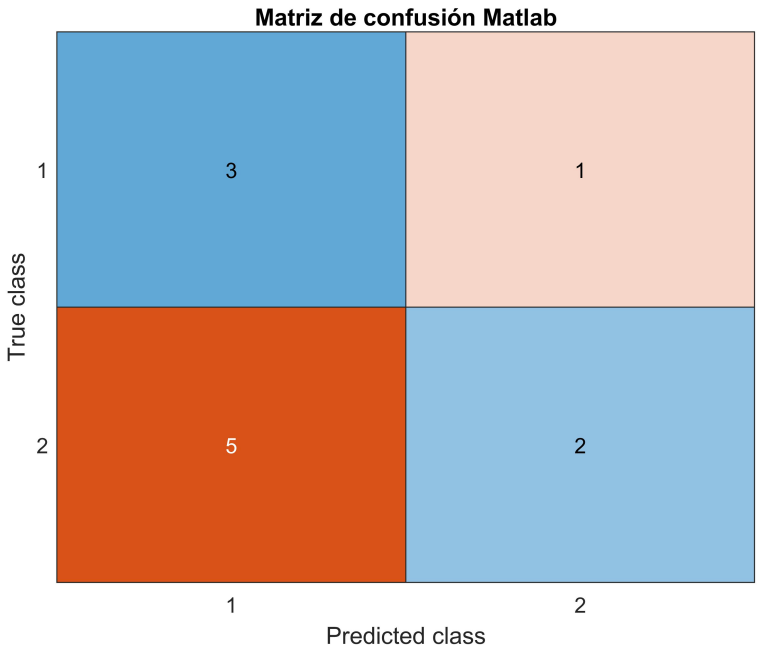

Fig. 27: Confusion matrix Matlab.

```
'''METRICAS DE ERROR'''
# Error cuadratico medio
mse = mean_squared_error(knn_predictions, te_class)

# CURVA ROC
fpr,tpr,tresholds = roc_curve(te_class, knn_predictions, pos_label=1)
plt.plot(fpr,tpr)
plt.xlabel('Ratio de falsos positivos')
plt.ylabel('Ratio de falsos negativos')
plt.show()

# MATRIZ DE CONFUSION
matriz = cm(te_class,knn_predictions)
VN,FP,FN,VP = cm(te_class,knn_predictions).ravel()
```

Listing 61: Código Python para generar la curva ROC y la AUC.

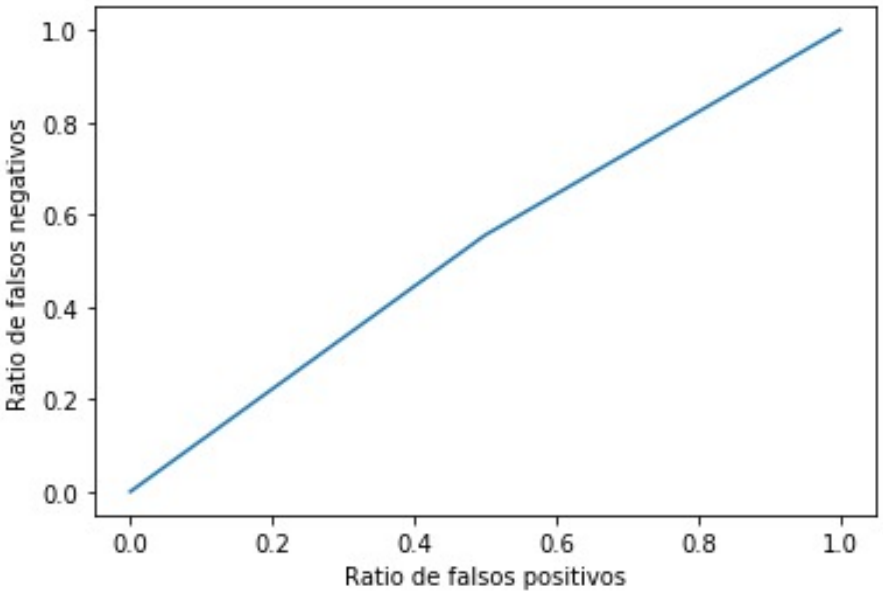

Fig. 28: ROC Python.

7 Discusión

A rasgos generales, las principales diferencias entre estos lenguajes son: En cuanto al tipo de software, Matlab es de pago; en cambio, Python y R son libres. Esto es un factor decisivo en la decisión de qué lenguaje utilizar. Matlab y R tienen muchas funciones integradas; mientras que en Python se han de programar o buscar bibliografía en blogs. En Python y R es necesario descargar las librerías a emplear antes de implementar funciones. Por último, para cada uno de los lenguajes podemos encontrar muchas de las funciones en sus respectivas páginas oficiales: MathWorks, Scikit-Learn y R Documentation.

7.1 Análisis exploratorio

A la hora de cargar los datos, podemos observar que con Matlab se pueden cargar directamente a través de un archivo '-.xlsx' mientras que con Python y R necesitamos un archivo "-.csv". En cuanto al histograma, la extensión del código es menor en Matlab, luego en R y por último en Python. Respecto a los resultados, no se aprecian diferencias entre los distintos lenguajes. Lo mismo ocurre con el kernel density, con boxplot y qqplot. Sin embargo, en Matlab, para la representación del boxplot y el qqplot, separemos las clases, tal y como se muestra en las figuras 9, 12 y 13. Las matrices de correlación en Python no presentan datos numéricos, únicamente representación gráfica. En R, los datos numéricos se representan en una matriz a parte, teniendo como resultado de la matriz de correlación dos gráficas, una de visualización y otra de datos numéricos. En Matlab, ambas gráficas se ven representadas en una sola figura. Los resultados son aproximadamente iguales.

7.2 Extracción de características

Para la extracción de características a partir de los métodos filters (Fscore y Relieff), hemos utilizado funciones integradas en todos los lenguajes, obteniendo como resultado para cada uno un ranking distinto. En cuanto a

los métodos Wrappers, en R y en Python obtenemos como resultado directamente las características relevantes. No obstante, en Matlab la función devuelve un vector de booleanos marcadas como 'True' las características más representativas. Cabe destacar, que en R es muy sencillo cambiar de método modificando los parámetros de la función y obtenemos una respuesta visualmente más atractiva. Respecto al Análisis de las Componentes Principales, destacar que en Python y en R directamente la función te devuelve la varianza explicada por cada componente y la suma acumulada de dichas varianzas. Mientras que en Matlab realizamos la gráfica pareto para obtener la suma acumulada. En R y Python es necesario desarrollar un código para implementar la gráfica pareto. De la misma forma, ha sido necesario desarrollar un código para implementar la función biplot en Python, a diferencia de R, donde existe una función dentro de la cuál podemos seleccionar como parámetro 'biplot' y Matlab, donde hay una función directa para ello.

7.3 Modelos de clasificación

Analizando los modelos de clasificación en Python, concluimos que la validación cruzada óptima es la que utiliza el método KFold con una media de 0.75 y una desviación típica de 0.10. Aunque a priori con el método LeaveOneOut se obtenga una media del score mayor (0.76), su desviación típica es demasiado elevada. En R se hace una partición del 30iteraciones, sacando la media de porcentaje de aciertos y la desviación típica del método.

En Matlab lo hacemos de la misma forma que en R, pero mediante el método HoldOut. Obteniendo como resultado la media, desviación típica, y los intervalos de confianza para la media y la desviación típica. Además de HoldOut en Matlab también conseguimos realizar particiones mediante los métodos LeaveOneOut y KFold cambiando el método en la función cvpartition. Cabe destacar que Matlab tiene optimización de hiperparámetros para mejorar los resultados de entrenamiento en función de los datos de entrada. Se ha realizado un gráfico en R para demostrar qué método es más eficiente en el entrenamiento de nuestros datos, tal y como muestra la figura 24.