Sistemas de Información y Telemedicina. *

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

January 16, 2020

Contents

Sección **Página**

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

List of Figures

Listings

1 Preámbulo

```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.

2 Histogramas

En este apartado dibujamos los histogramas comparativos.

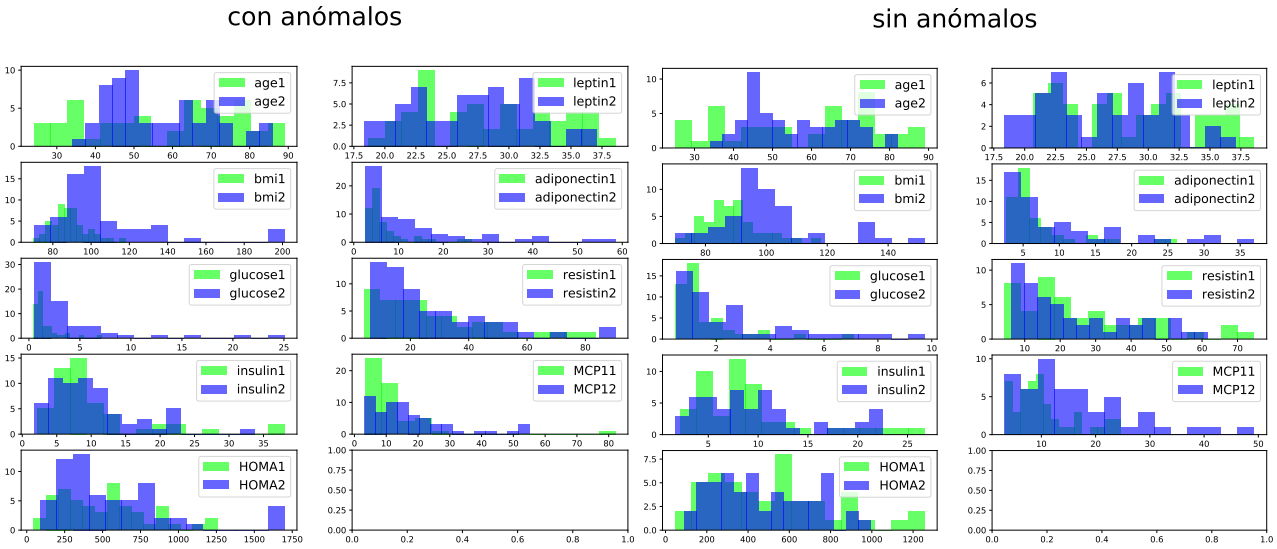

Fig. 1: Histogramas Python para datos con y sin anomalias.

```
1  import matplotlib as mpl
2  import matplotlib.pyplot as plt
3
4  #  load preprocessed data, x and y are raw, x_no and y_no contain no outliers
5  from preprocessing import x, y, x_no, y_no, labels
6
7  #  colours for the histograms
8  fc = [(), (0, 1, 0, 0.6), (0, 0, 1, 0.6)]
9  #          (R, G, B,  α )← transparency
10
11 fig, ax = plt.subplots (nrows = 5, ncols = 2, figsize = (13, 10))
12 ax = ax.flatten ()
13
14 #  draws each of the histograms, two for each variable
15 for i in range (0, 9):
16     for j in [1, 2]:
17         ax[i].hist (x [y == j, i], bins = 15, fc = fc [j], label = labels [i] + str ↘
               (j))
18         ax[i].legend (loc = 1, prop={'size': 15})
19
20 fig.suptitle ('con anómalos', fontsize = 30)
21 fig.savefig  ('../images/hist.pdf', bbox_inches = 'tight', pad_inches = 0)
```

Listing 3: Código Python generador de los histogramas con datos anómalos.

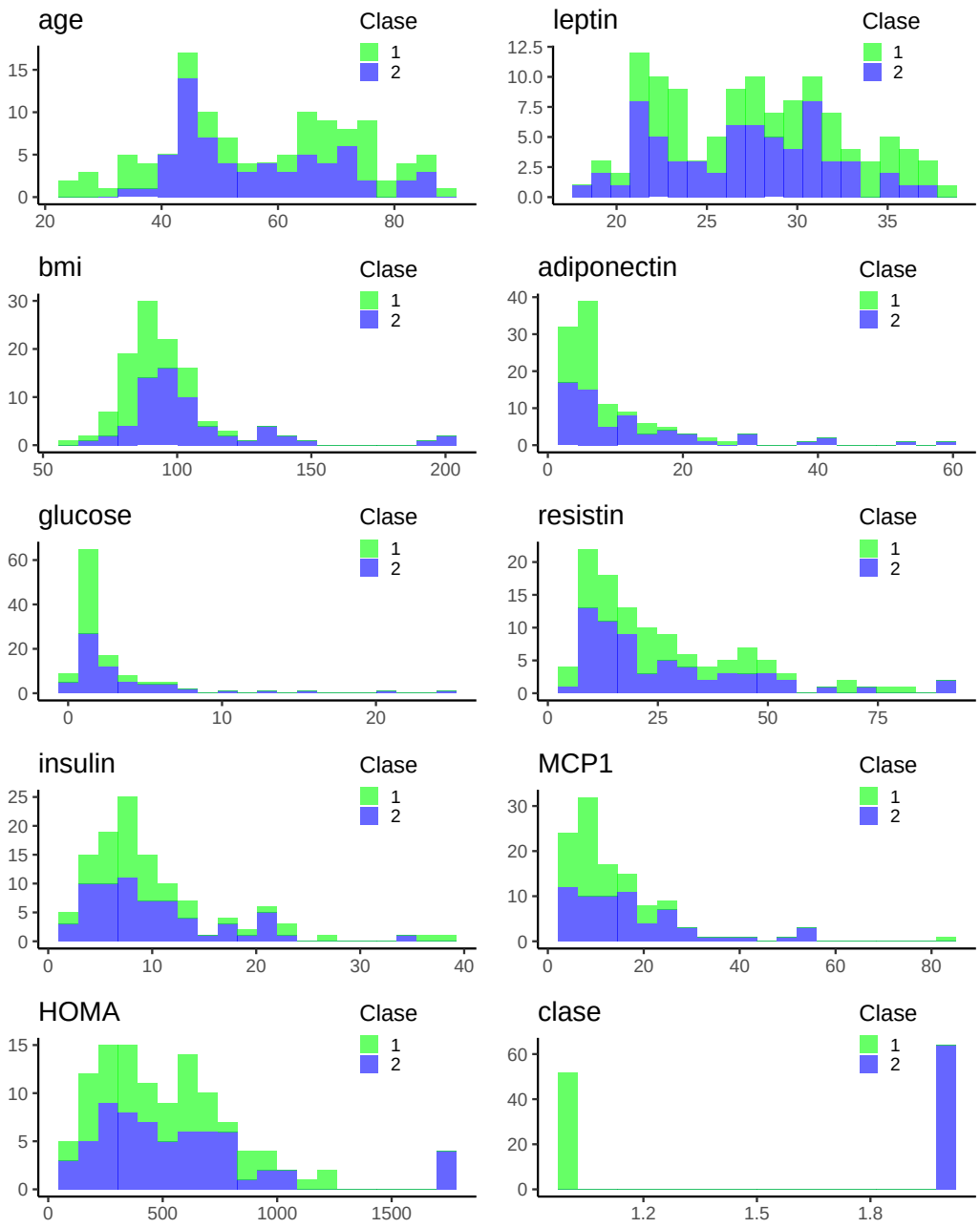

Fig. 2: Histogramas R para datos con anomalias.

```r
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 4: Código R generador de los histrogramas con datos anómalos.

3 Kernel Density

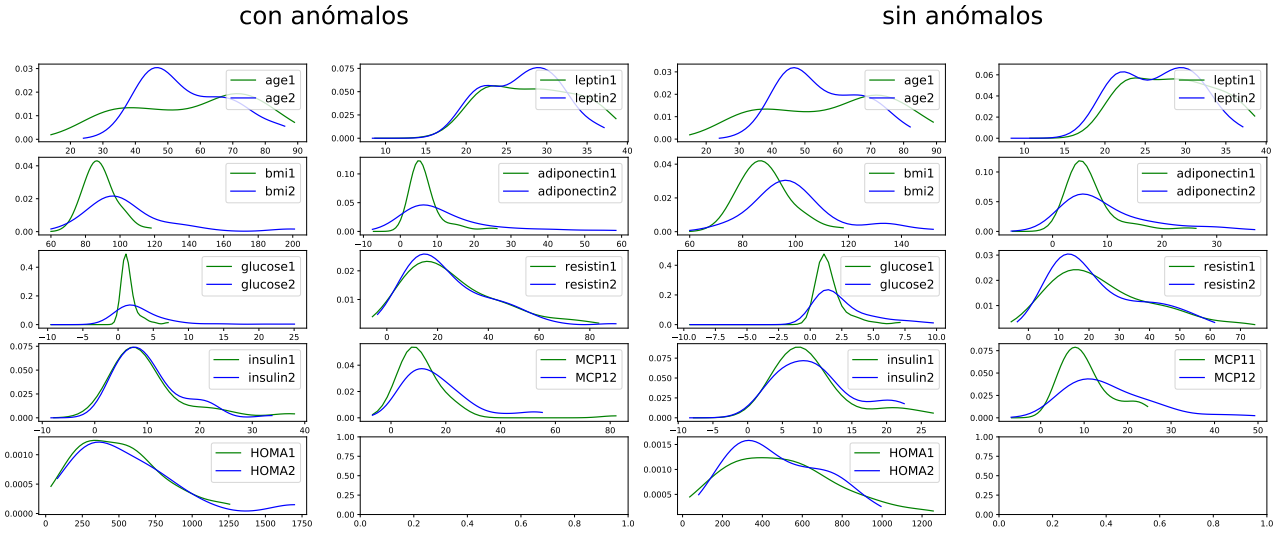

Fig. 3: Kernel Density para datos con y sin anomalias.

```python
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 5: Código Python generador de los kernel density plots con datos anómalos.

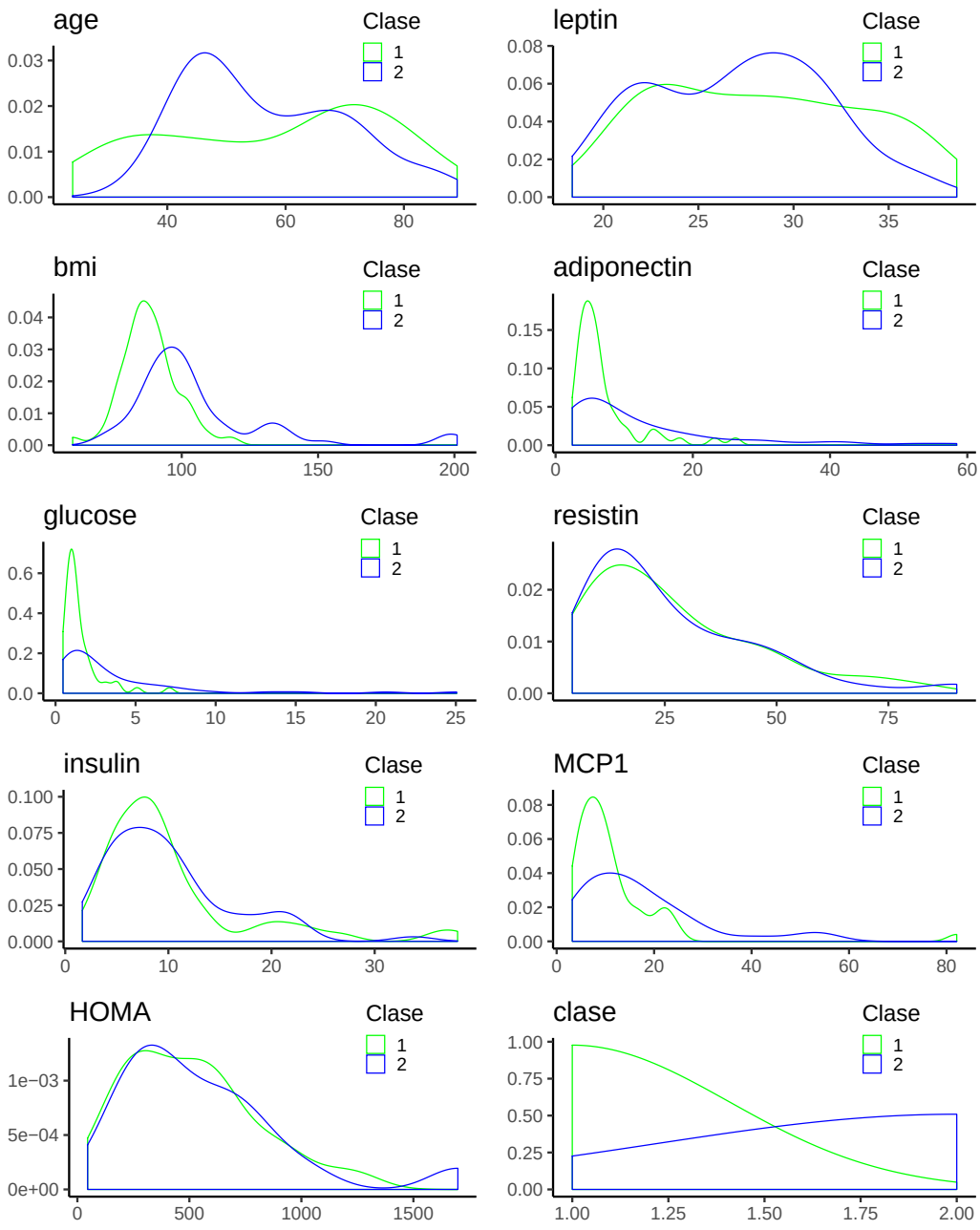

Fig. 4: Gráficos de densidad R.

```
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 6: Código R generador de los density plots.

4 Boxplot

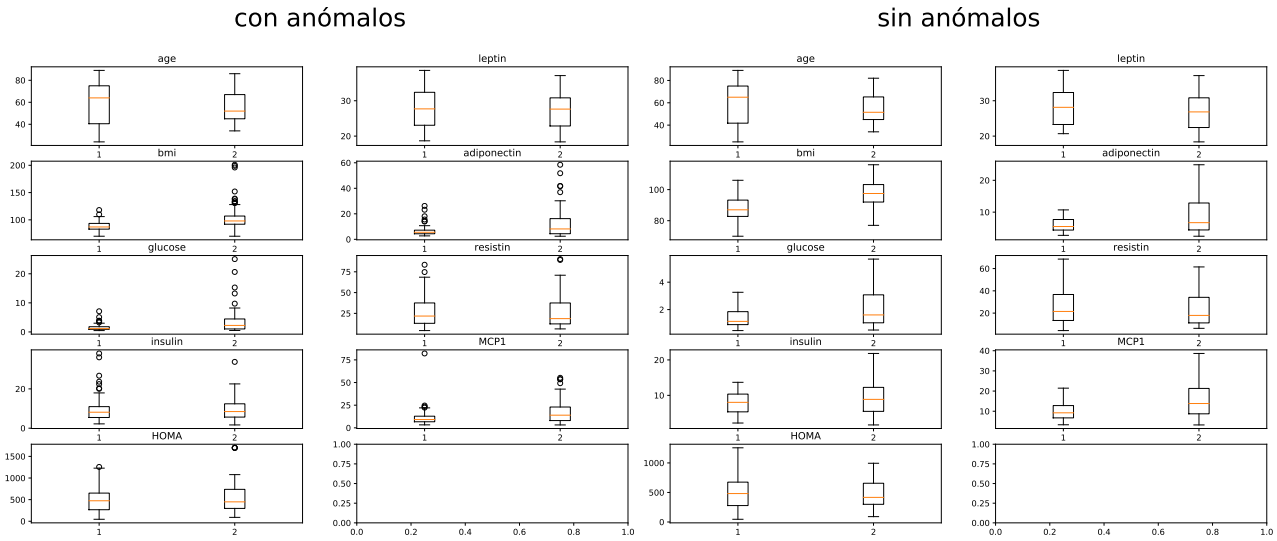

Fig. 5: Boxplots Python para datos con y sin anomalias.

```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 7: Código Python generador de los boxplots con datos anómalos.

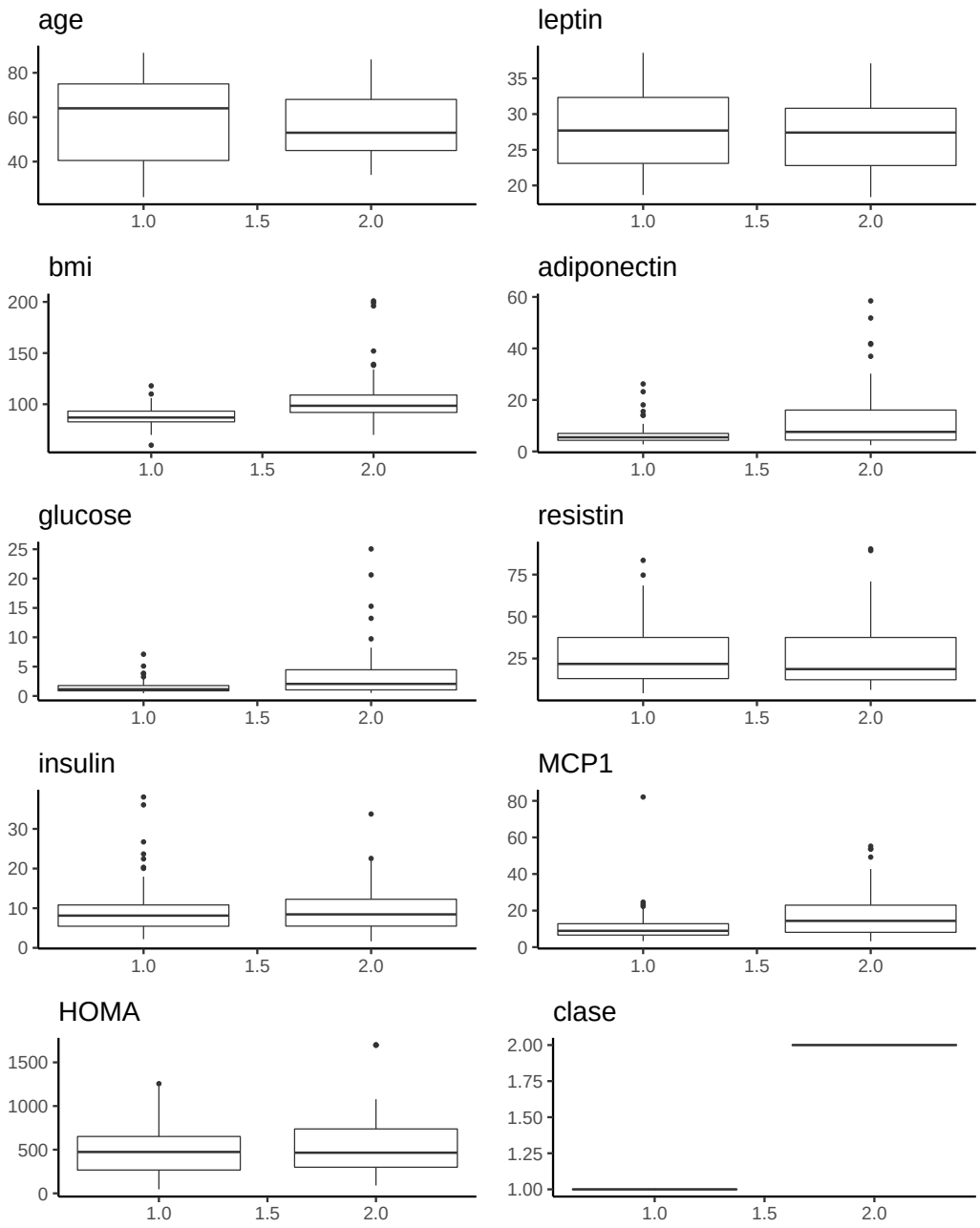

Fig. 6: Boxplots R para datos con anomalias.

```
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 8: Código R generador de los boxplots con datos anómalos.

5 QQplot

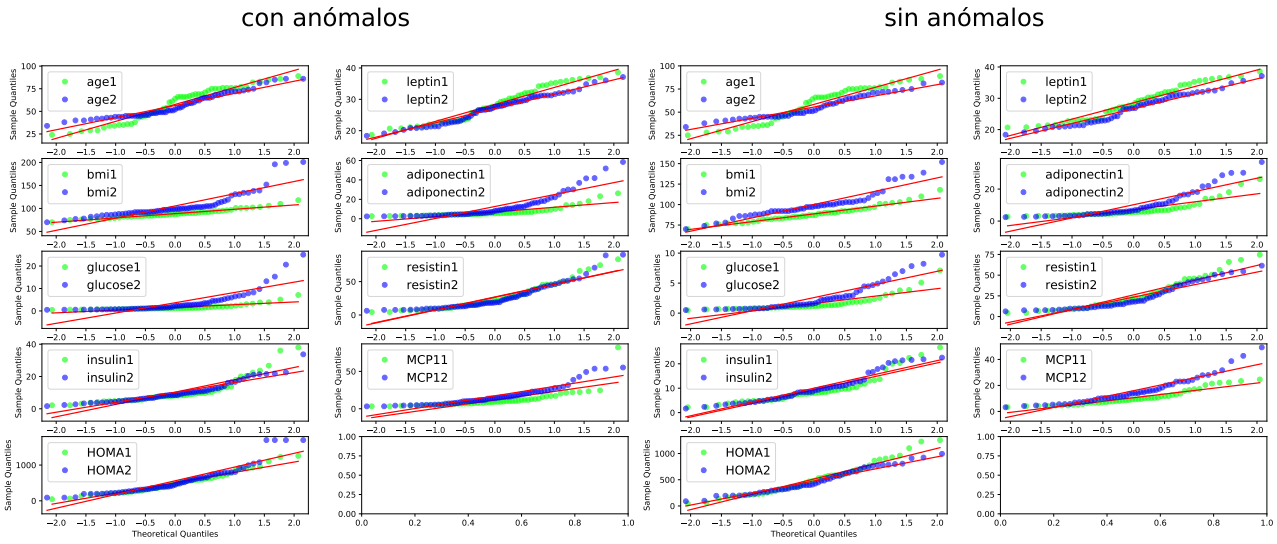

Fig. 7: QQplots Python para datos con y sin anomalias.

```
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 9: Código Python generador de los QQplots con datos anómalos.

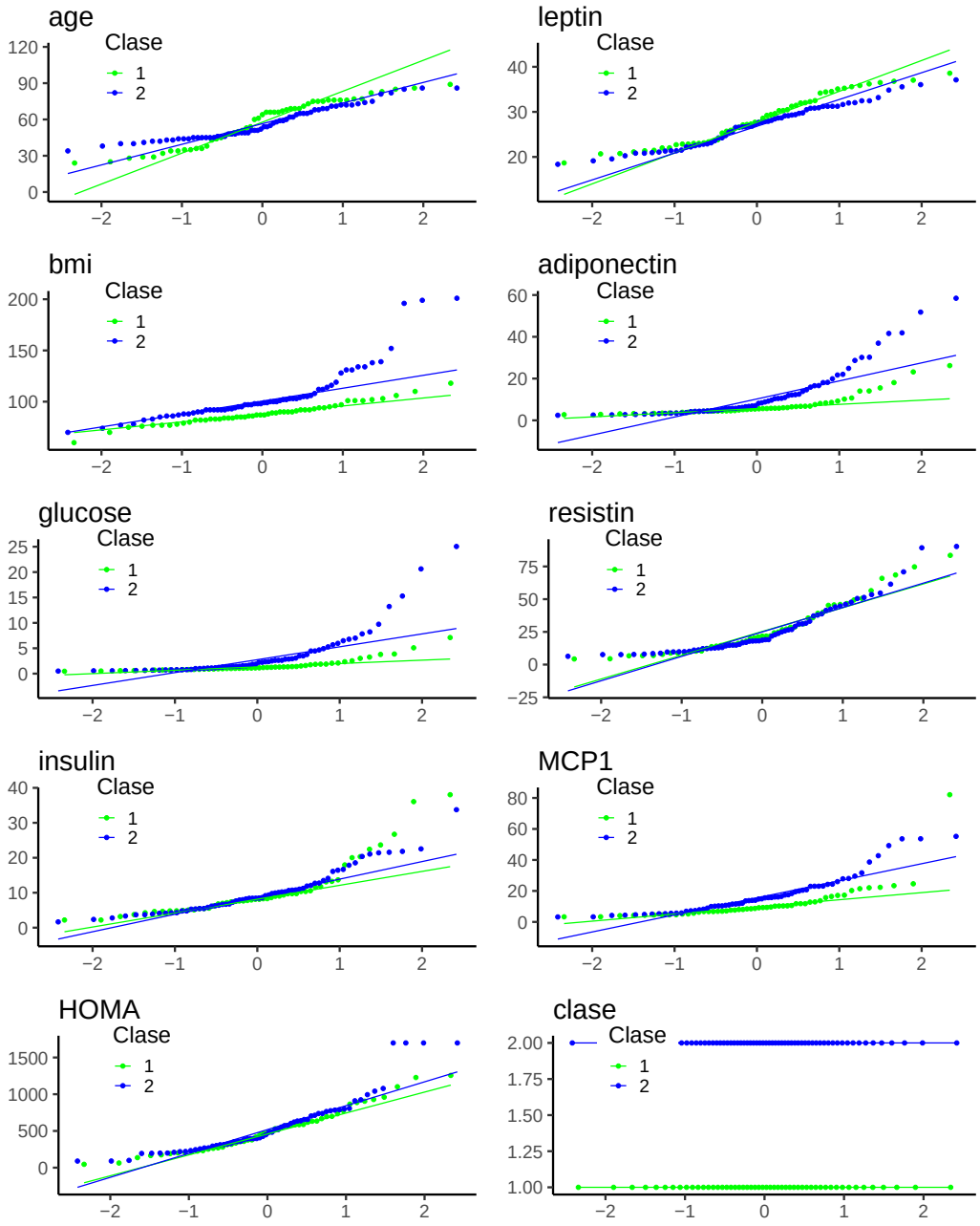

Fig. 8: QQplots R.

```
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 10: Código R generador de los QQplots.

6 Corrplot

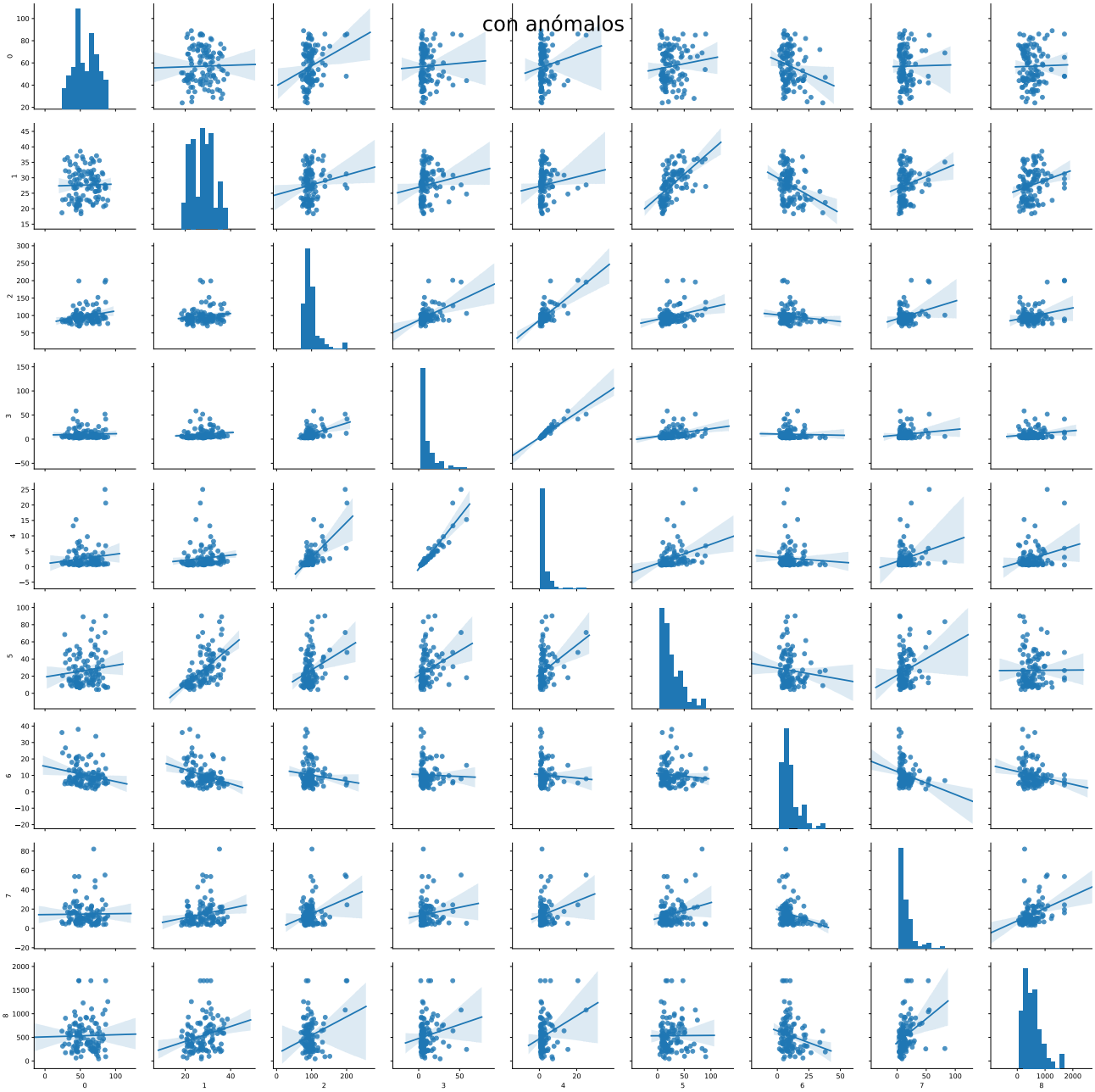

Fig. 9: Corrplot Python para datos con anomalias.

```
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 11: Código Python generador de los corrplots con datos anómalos.

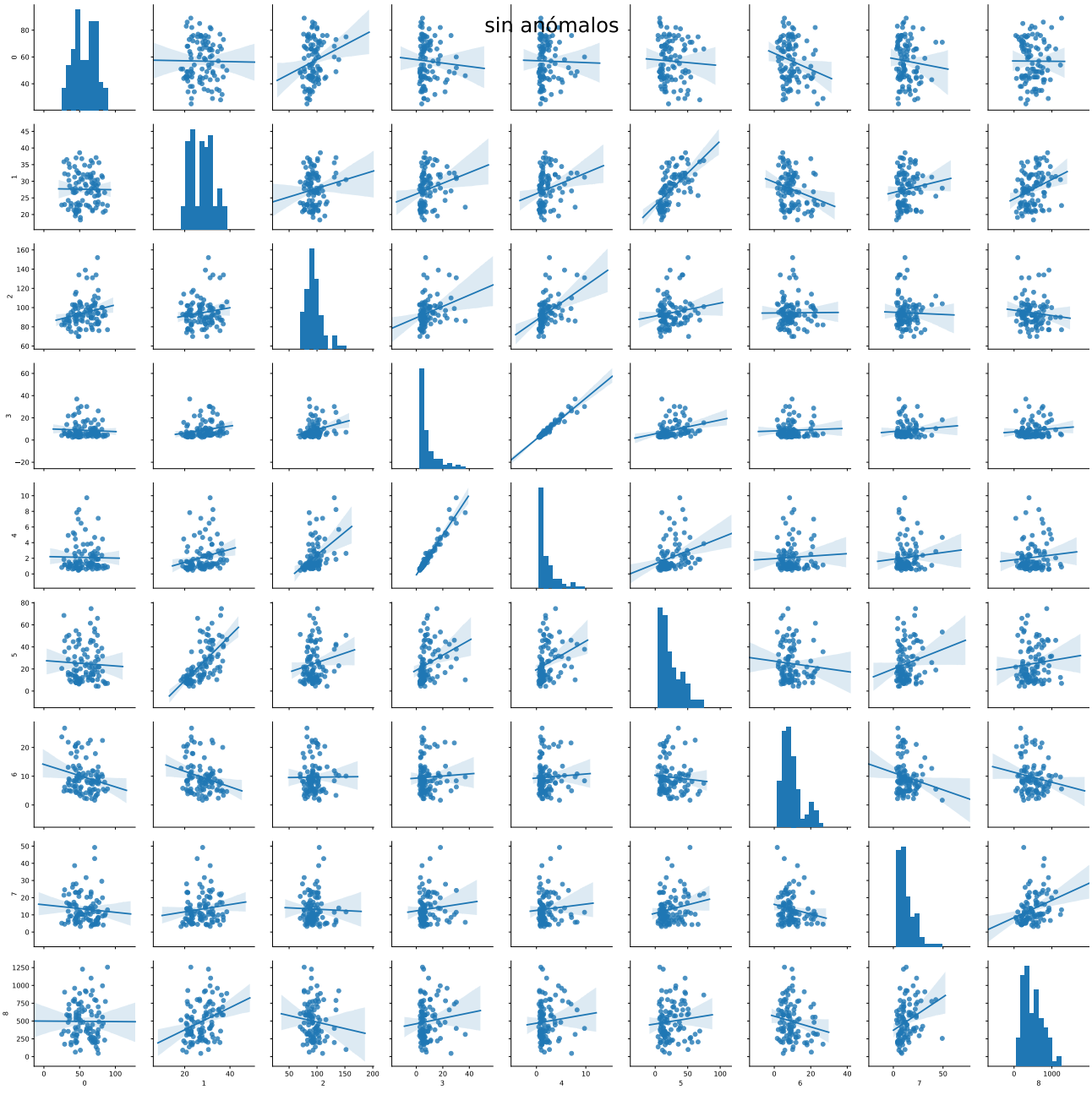

Fig. 10: Corrplot Python para datos sin anomalias.

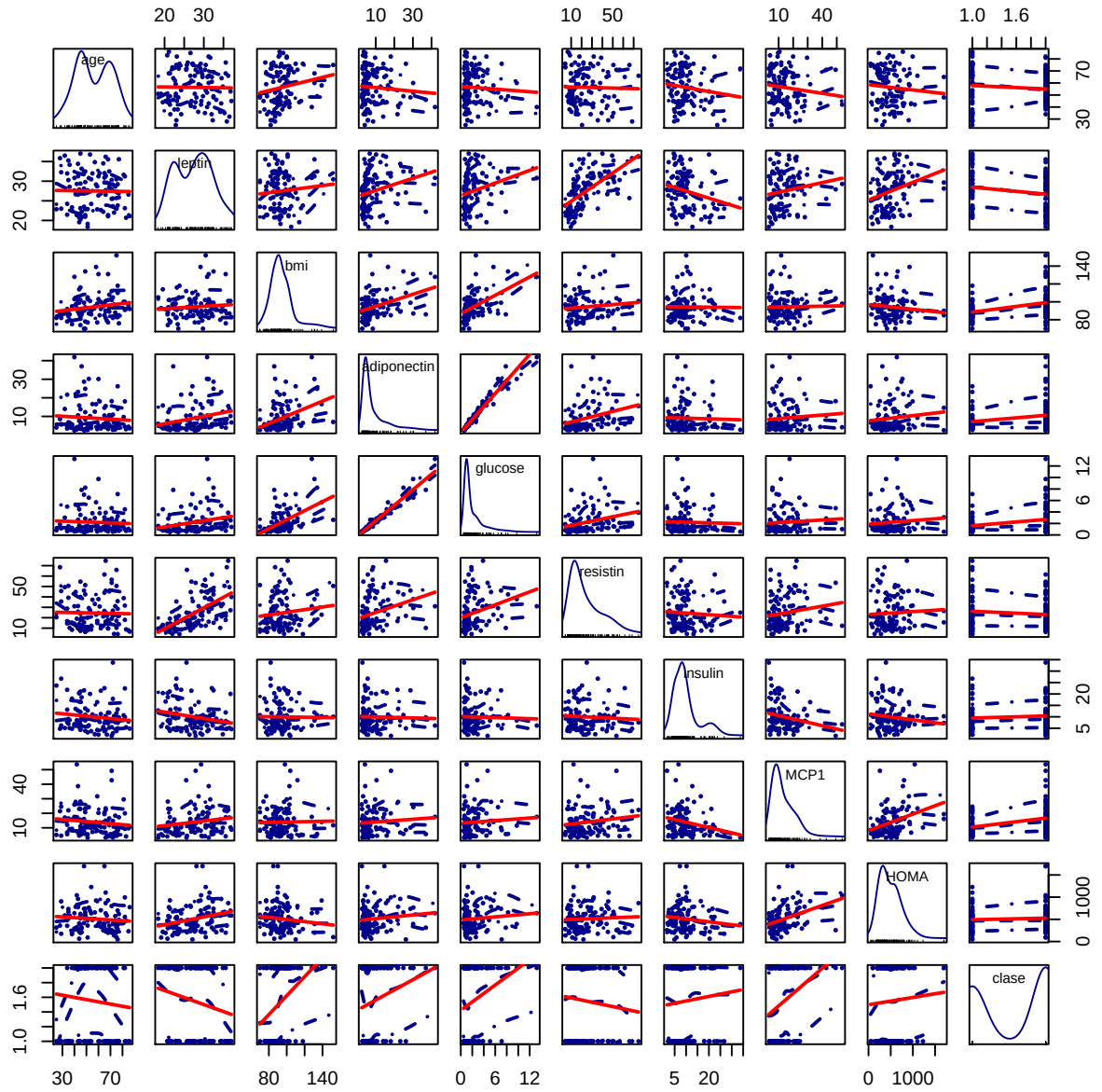

Fig. 11: Corrplot R para datos con anomalias.

```
library (car)
pdf ("../images/corrplot.pdf")
scatterplotMatrix (datos, regLine=list (col='red'), pch=20, cex=0.5, col='blue4')
dev.off ()

library (corrplot)
pdf ("../images/corrplot1.pdf")
M <- cor (na.omit (datos))
corrplot (M, method = 'number')
dev.off ()
```

Listing 12: Código R generador de los corrplots.

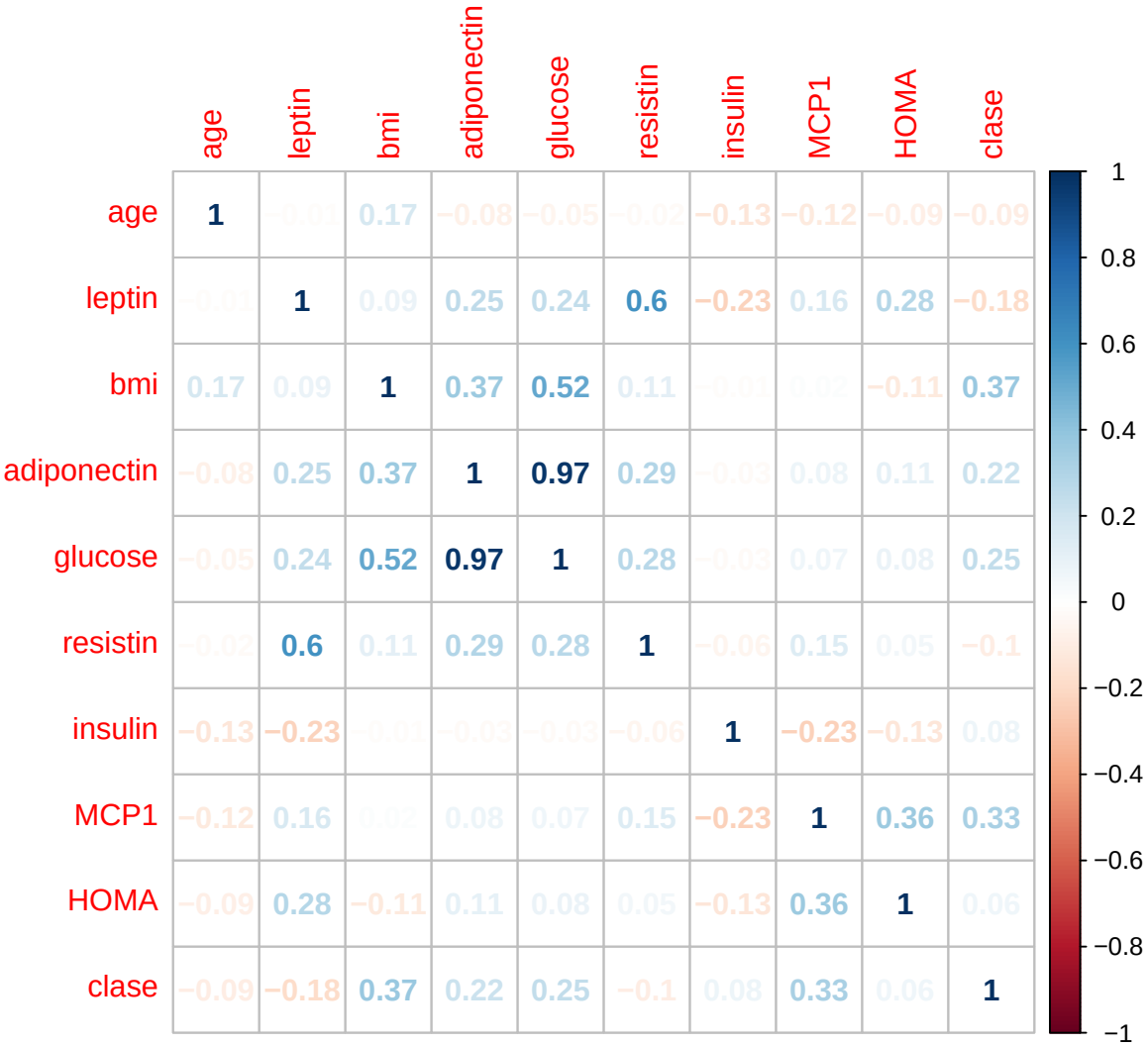

Fig. 12: Matriz de correlaciones en R.

7 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 13: Aplicación métodos *filter* de selección características.

```
[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 14: Ranking de variables según los métodos filter.

```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 15: Ranking de variables según distintos métodos en R.

8 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 16: 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 17: Resultados Python del filtrado mediante wrappers.

```
# 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 18: Resultados R del filtrado mediante wrappers.

```
# esto es extra
library (Boruta)
Boruta (as.factor (clase)~., datos, maxRuns = 101) -> borutaout

# Boruta performed 100 iterations in 4.317041 secs.
#  5 attributes confirmed important: age, bmi, glucose, leptin, MCP1;
#  3 attributes confirmed unimportant: HOMA, insulin, resistin;
#  1 tentative attributes left: adiponectin;

pdf ("../images/boruta.pdf")
plot (borutaout, las = 2, xlab = '', main = 'Boruta Variable Importance')
dev.off ()
```

Listing 19: Método Boruta *wrapper* de Random Forest R.

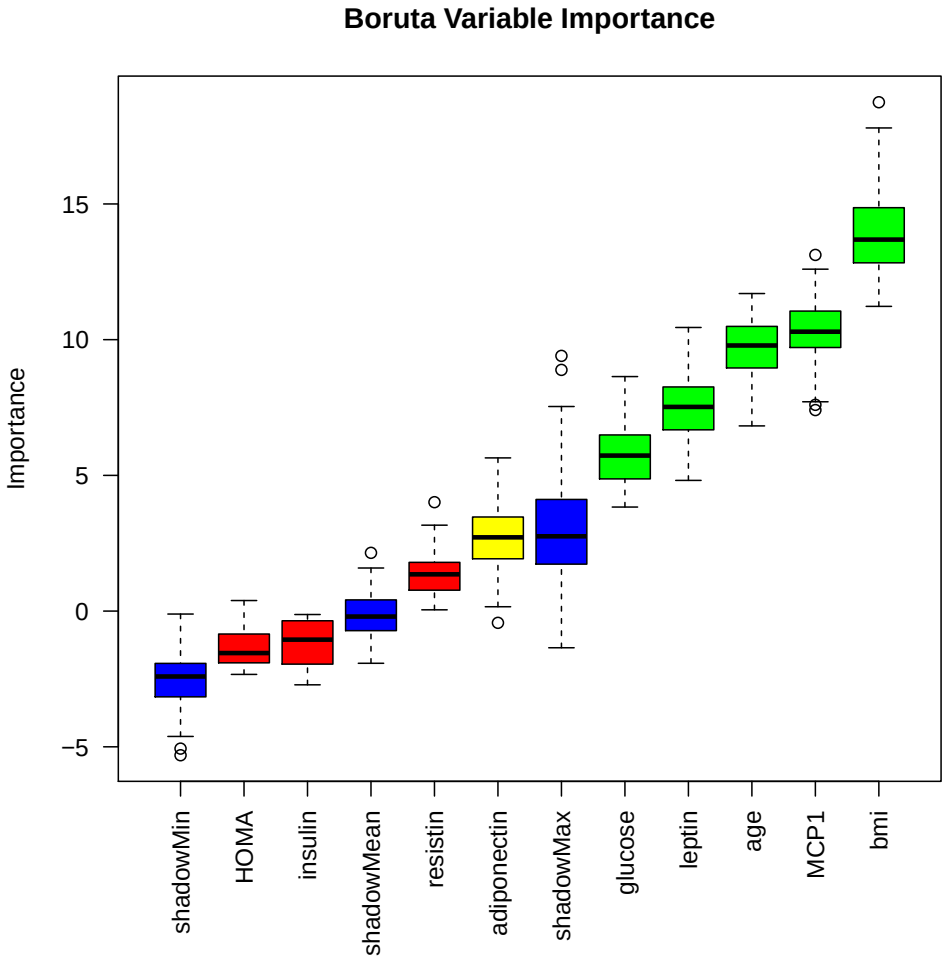

Fig. 13: Representación gráfica de la importancia de las variables seleccionadas por Boruta.

9 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 20: *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 21: Varianza explicada por componente y suma acumulada Python.

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

Listing 22: *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 23: Varianza explicada por componente y suma acumulada R.

9.1 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 24: Código generador del diagrama de Pareto en Python.

```r
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 25: Código generador del diagrama de Pareto en R.

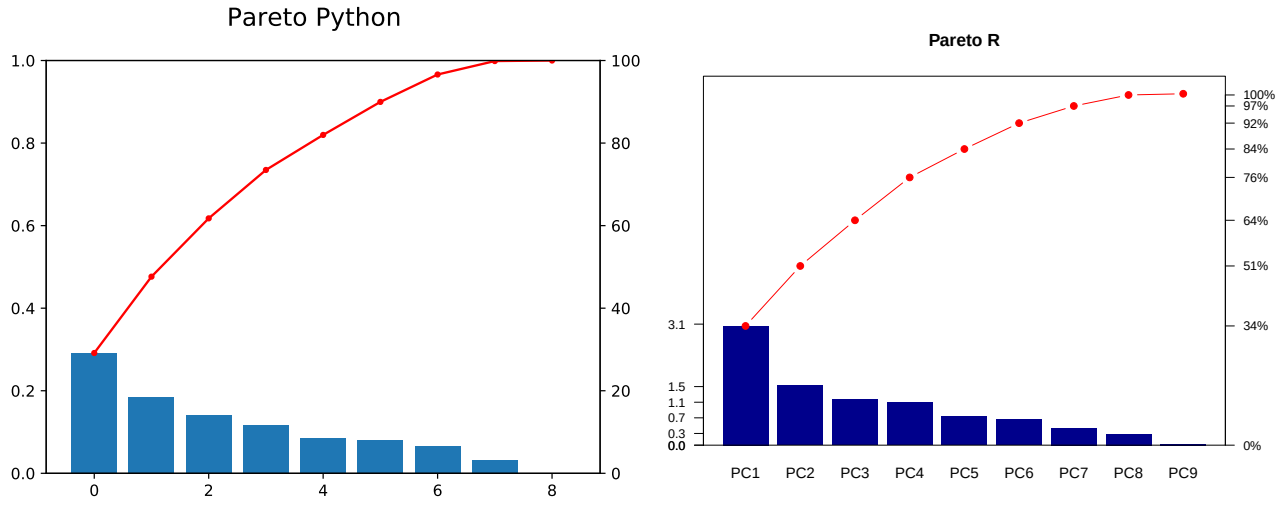

Fig. 14: Diagrama de Pareto en Python y R.

9.2 Biplot

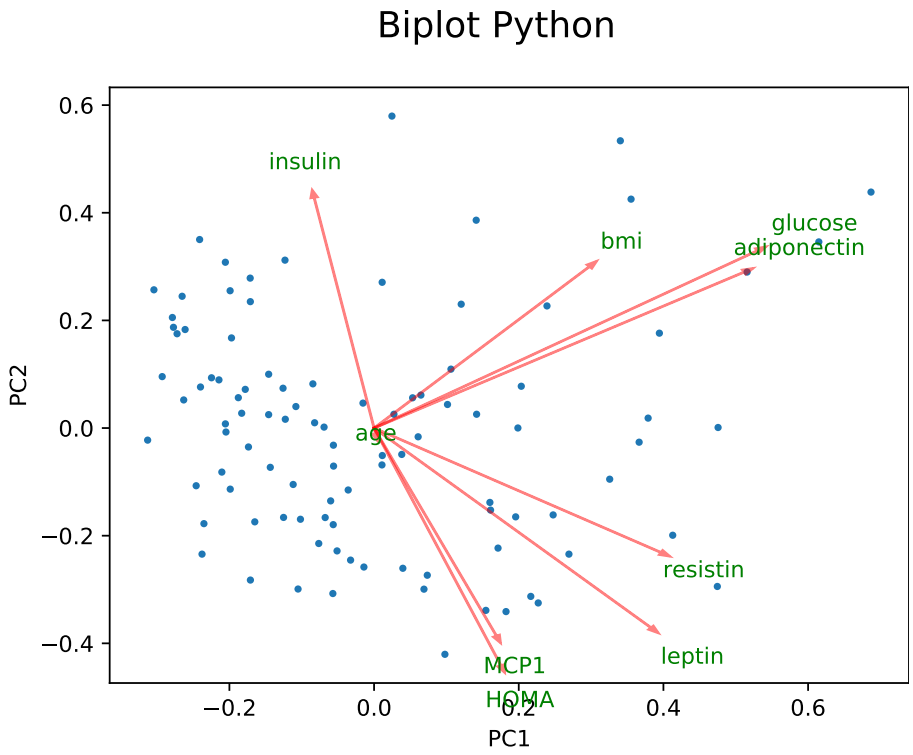

Fig. 15: Biplot Python.

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

Listing 26: Código generador del Biplot en Python.

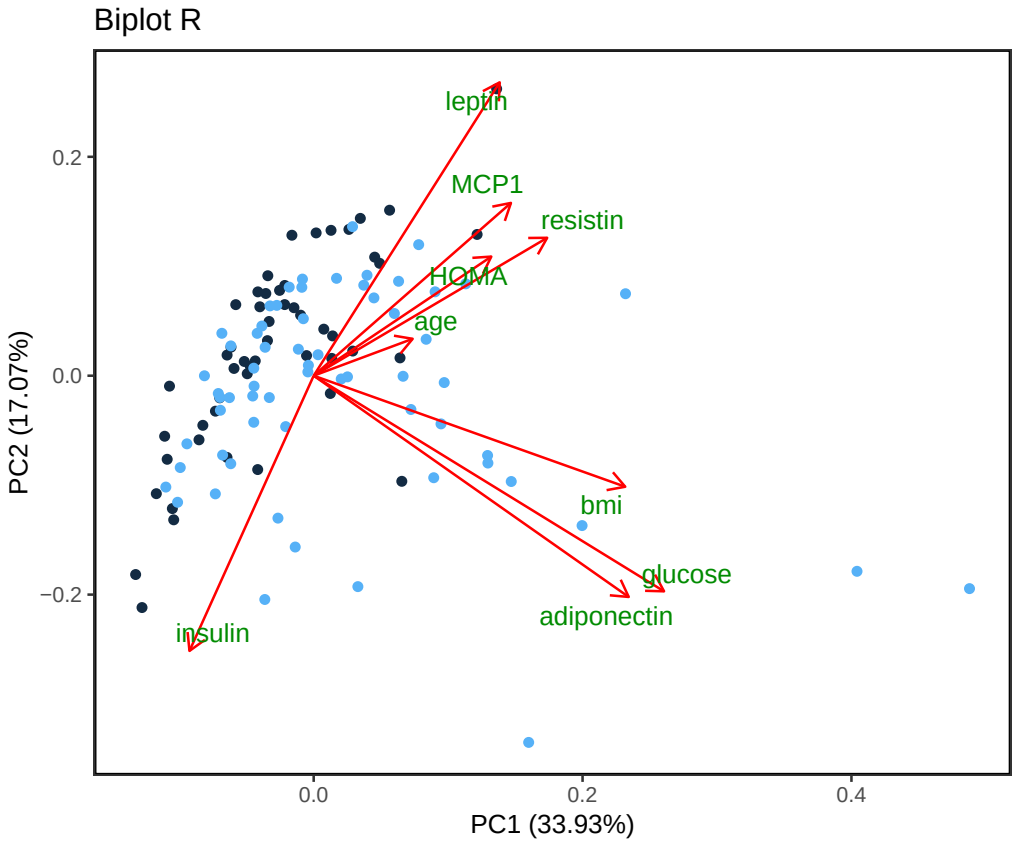

Fig. 16: Biplot R.

```
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 27: Código generador del Biplot en R.

10 Modelos de Clasificación

10.1 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 28: 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 29: 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 30: 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 31: R puntuación de mil evaluaciones.

10.2 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 32: 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 33: 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 34: 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 35: R puntuación de mil evaluaciones.

10.3 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 36: 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 37: 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 38: 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 39: R puntuación de mil evaluaciones.

11 Postámbulo

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

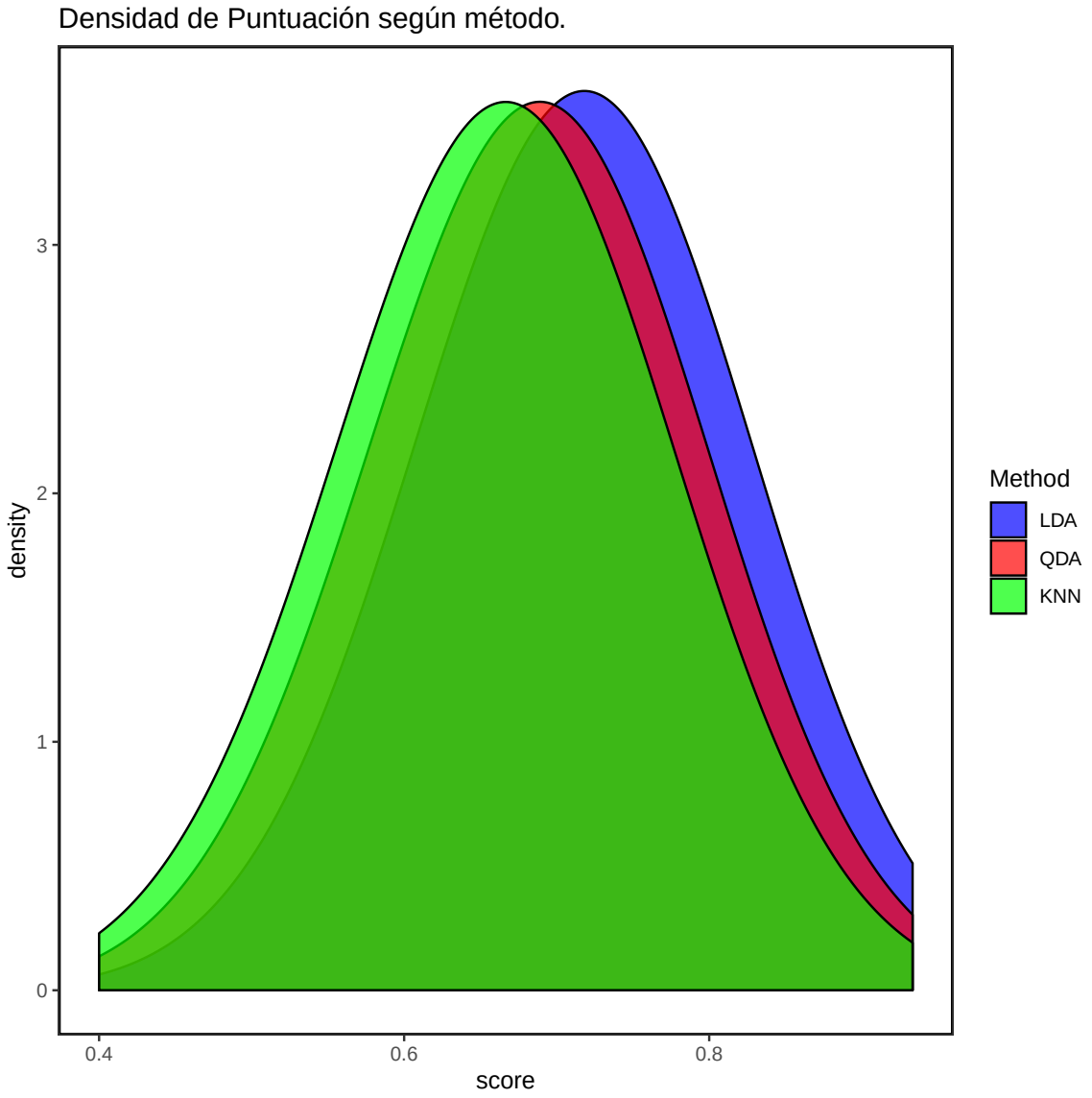

Fig. 17: Comparación de mil evaluaciones de cada uno de los métodos de clasificación en R.

11.2 Puntuación *knn* vs. número de vecinos.

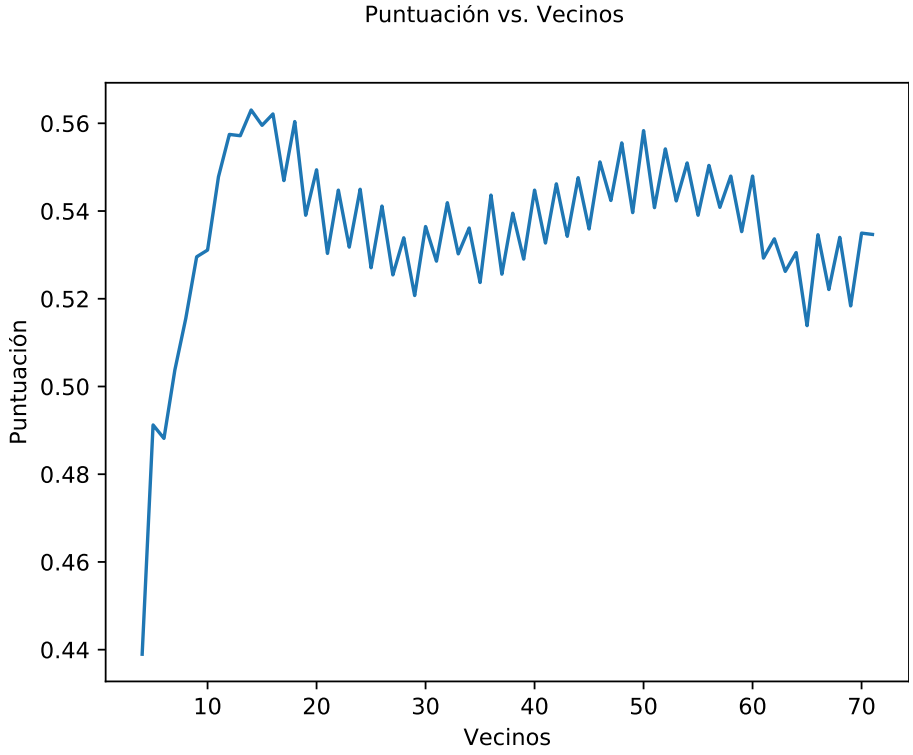

Fig. 18: Rendimineto decreciente según aumenta el número de vecinos.

```
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 40: Evolución de puntuación según número de vecinos.

11.3 *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 41: 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 42: 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 43: 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 44: Tiempo de ejecución en Python.