

Sistemas de Información y Telemedicina. *

Marta Girones Sanguesa

Silvia Marset Gomis

Ignacio Amat Hernández

Sofía Gutiérrez Santamaría

January 2, 2020

Contents

| Sección | Página |
|--|-----------|
| 1 Preámbulo | 4 |
| 2 Histogramas | 5 |
| 3 Kernel Density | 7 |
| 4 Boxplot | 9 |
| 5 QQplot | 11 |
| 6 Corrplot | 13 |
| 7 Filter Methods | 17 |
| 8 Wrapper Methods | 19 |
| 9 PCA | 22 |
| 9.1 Pareto | 22 |
| 9.2 Biplot | 24 |
| 10 Modelos de Clasificación | 26 |
| 10.1 Clasificación Lineal | 26 |
| 10.2 Clasificación Cuadrática | 28 |
| 10.3 Clasificación KNN | 30 |
| 11 Postámbulo | 32 |

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

List of Figures

| | | |
|----|---|----|
| 1 | Histogramas Python para datos con y sin anomalías. | 5 |
| 2 | Histogramas R para datos con anomalías. | 6 |
| 3 | Kernel Density para datos con y sin anomalías. | 7 |
| 4 | Gráficos de densidad R. | 8 |
| 5 | Boxplots Python para datos con y sin anomalías. | 9 |
| 6 | Boxplots R para datos con anomalías. | 10 |
| 7 | QQplots Python para datos con y sin anomalías. | 11 |
| 8 | QQplots R. | 12 |
| 9 | Corrplot Python para datos con anomalías. | 13 |
| 10 | Corrplot Python para datos sin anomalías. | 14 |
| 11 | Corrplot R para datos con anomalías. | 15 |
| 12 | Matriz de correlaciones en R. | 16 |
| 13 | Representación gráfica de la importancia de las variables seleccionadas por Boruta. | 21 |
| 14 | Diagrama de Pareto en Python y R. | 23 |
| 15 | Biplot Python. | 24 |
| 16 | Biplot R. | 25 |
| 17 | Rendimiento decreciente según aumenta el número de vecinos. | 32 |
| 18 | Comparación de mil evaluaciones de cada uno de los métodos de clasificación en R. | 33 |

Listings

| | | |
|----|---|----|
| 1 | Importaciones iniciales y preparación de datos en Python. | 4 |
| 2 | Importaciones iniciales y preparación de datos en R. | 4 |
| 3 | Código Python generador de los histogramas con datos anómalos. | 5 |
| 4 | Código R generador de los histogramas con datos anómalos. | 6 |
| 5 | Código Python generador de los kernel density plots con datos anómalos. | 7 |
| 6 | Código R generador de los density plots. | 8 |
| 7 | Código Python generador de los boxplots con datos anómalos. | 9 |
| 8 | Código R generador de los boxplots con datos anómalos. | 10 |
| 9 | Código Python generador de los QQplots con datos anómalos. | 11 |
| 10 | Código R generador de los QQplots. | 12 |
| 11 | Código Python generador de los corrplots con datos anómalos. | 13 |
| 12 | Código R generador de los corrplots. | 15 |
| 13 | Aplicación métodos <i>filter</i> de selección características. | 17 |
| 14 | Ranking de variables según los métodos filter. | 17 |
| 15 | Ranking de variables según distintos métodos en R. | 18 |

| | | |
|----|---|----|
| 16 | Aplicación métodos <i>wrapper</i> de selección características. | 19 |
| 17 | Resultados Python del filtrado mediante wrappers. | 19 |
| 18 | Resultados R del filtrado mediante wrappers. | 20 |
| 19 | Método Boruta <i>wrapper</i> de Random Forest R. | 20 |
| 20 | <i>Principal Component Analysis</i> Python. | 22 |
| 21 | Varianza explicada por componente y suma acumulada Python. | 22 |
| 22 | <i>Principal Component Analysis</i> R. | 22 |
| 23 | Varianza explicada por componente y suma acumulada R. | 22 |
| 24 | Código generador del diagrama de Pareto en Python. | 22 |
| 25 | Código generador del diagrama de Pareto en R. | 23 |
| 26 | Código generador del Biplot en Python. | 24 |
| 27 | Código generador del Biplot en R. | 25 |
| 28 | Python validación del modelo lineal. | 26 |
| 29 | Python validación según distintos métodos de partición. | 26 |
| 30 | R análisis lineal discriminante. | 27 |
| 31 | R puntuación de mil evaluaciones. | 27 |
| 32 | Python validación del modelo cuadrático. | 28 |
| 33 | Python validación según distintos métodos de partición. | 28 |
| 34 | R análisis cuadrático discriminante. | 29 |
| 35 | R puntuación de mil evaluaciones. | 29 |
| 36 | Python validación del modelo KNN. | 30 |
| 37 | Python validación según distintos métodos de partición. | 30 |
| 38 | R <i>K nearest neighbours</i> | 31 |
| 39 | R puntuación de mil evaluaciones. | 31 |
| 40 | Evolución de puntuación según número de vecinos. | 32 |

1 Preámbulo

```
1 import numpy as np
2 from scipy import stats
3
4 # names of variables
5 labels = ['age', 'leptin', 'bmi', 'adiponectin', 'glucose',
6           'resistin', 'insulin', 'MCP1', 'HOMA']
7
8 # loads data
9 data = np.loadtxt (open (r'../data.csv', 'rb'), delimiter = ',', skiprows = 1)
10
11 # rewrites data as all the rows of data w/out nan cells
12 data = data [~np.isnan (data).any (axis=1)]
13
14 # separates parameters into matrix x
15 x = np.array ([list (data [x][: -1]) for x in range (len (data))])
16
17 # and class (1, 2) into vector y
18 y = np.array ([int (data [x][ -1]) for x in range (len (data))])
19
20 # removes outliers
21 data_no = data [(np.abs (stats.zscore (data)) < 3).all (axis = 1)]
22 # ↑ = No Outliers
23
24 x_no = np.array ([list (data_no [x][: -1]) for x in range (len (data_no))])
25 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.

```
1 # load data
2 datos <- read.table ('../data.csv', sep = ',', header = T)
3 datos <- na.omit (datos)
4
5 # ignore rows w/ components above the 99th percentile
6 suppressPackageStartupMessages (library (dplyr))
7 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.

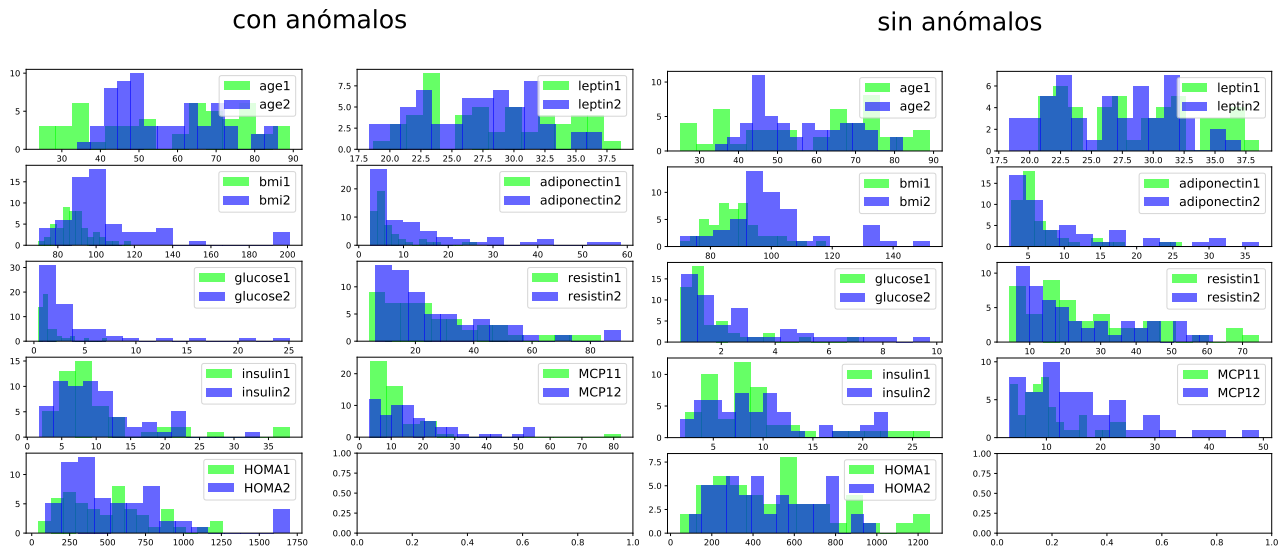


Fig. 1: Histogramas Python para datos con y sin anomalías.

```

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 \
18                     (j))
19         ax[i].legend (loc = 1, prop={'size': 15})
20
21 fig.suptitle ('con anómalos', fontsize = 30)
22 fig.savefig ('../images/hist.pdf', bbox_inches = 'tight', pad_inches = 0)

```

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

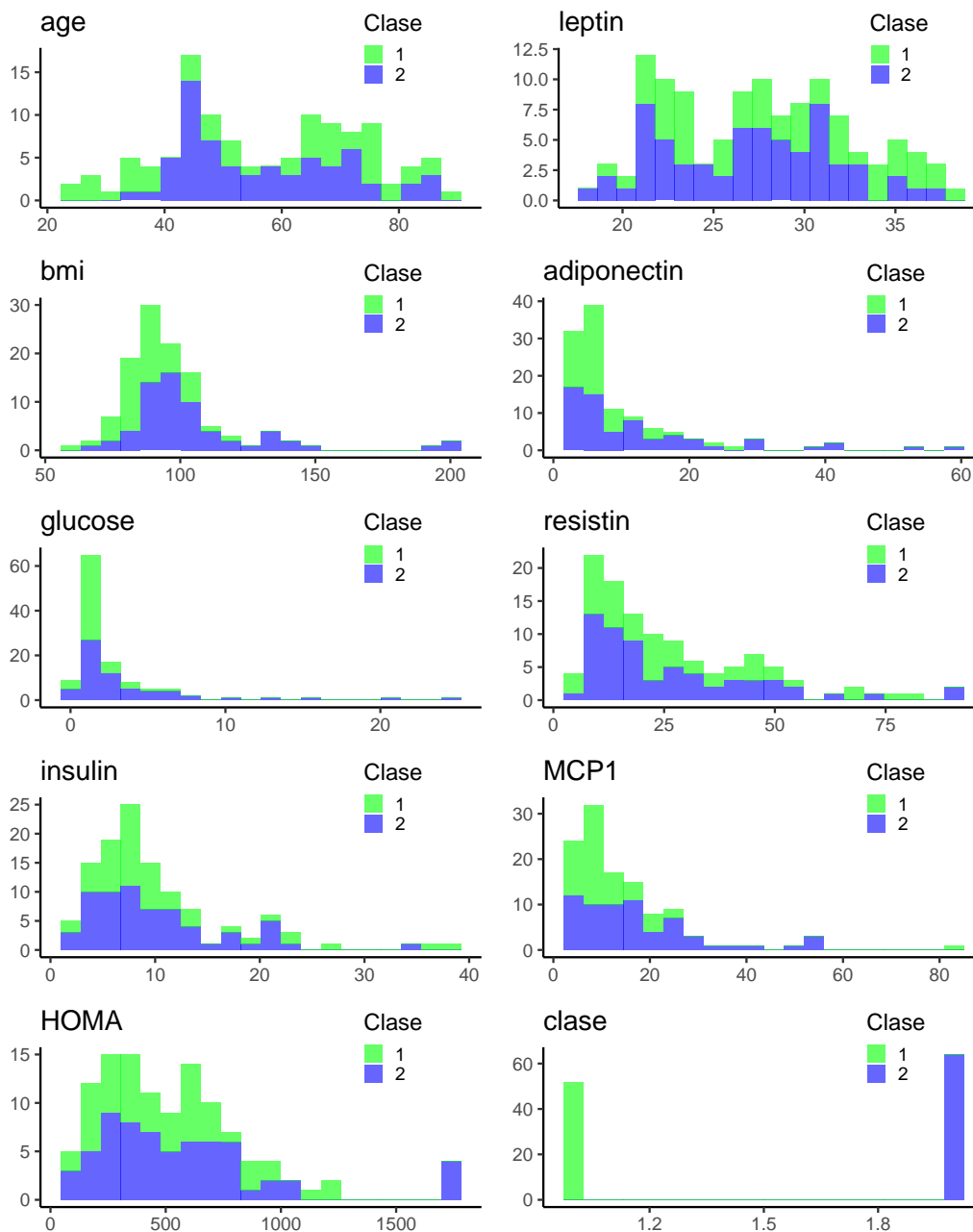


Fig. 2: Histogramas R para datos con anomalías.

```

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

```

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

3 Kernel Density

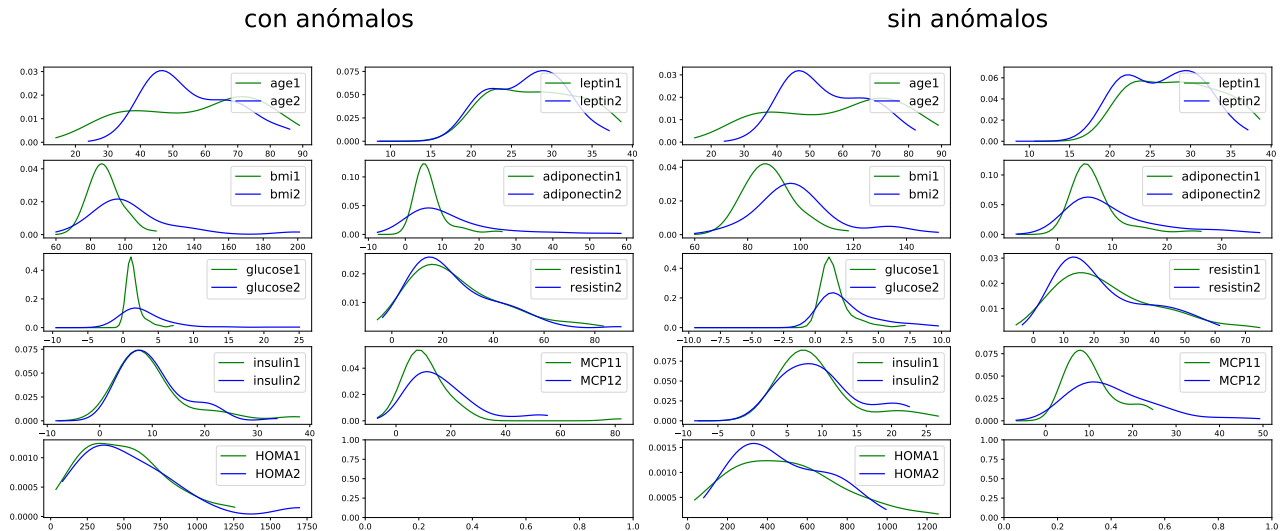


Fig. 3: Kernel Density para datos con y sin anomalias.

```

1  import matplotlib as mpl
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from scipy.stats import gaussian_kde
5
6  # load preprocessed data, x and y are raw, x_no and y_no contain no outliers
7  from preprocessing import x, y, x_no, y_no, labels
8
9  # colours
10 fc = ['', 'green', 'blue']
11
12 fig, ax = plt.subplots (nrows = 5, ncols = 2, figsize = (13, 10))
13 ax = ax.flatten ()
14
15 # same loop in principle as before
16 for i in range (0, 9):
17     for j in [1, 2]:
18         kde = gaussian_kde (x_ := x [y == j, i])
19         xs = np.linspace(np.min (x_) - 10, np.max (x_), num=len (x_))
20         ax[i].plot (xs, kde(xs), c = fc[j], label = labels [i] + str (j))
21         ax[i].legend (loc = 1, prop={'size': 15})
22
23 fig.suptitle ('con anomalos', fontsize = 30)
24 fig.savefig ('../images/kden.pdf', bbox_inches = 'tight', pad_inches = 0)

```

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

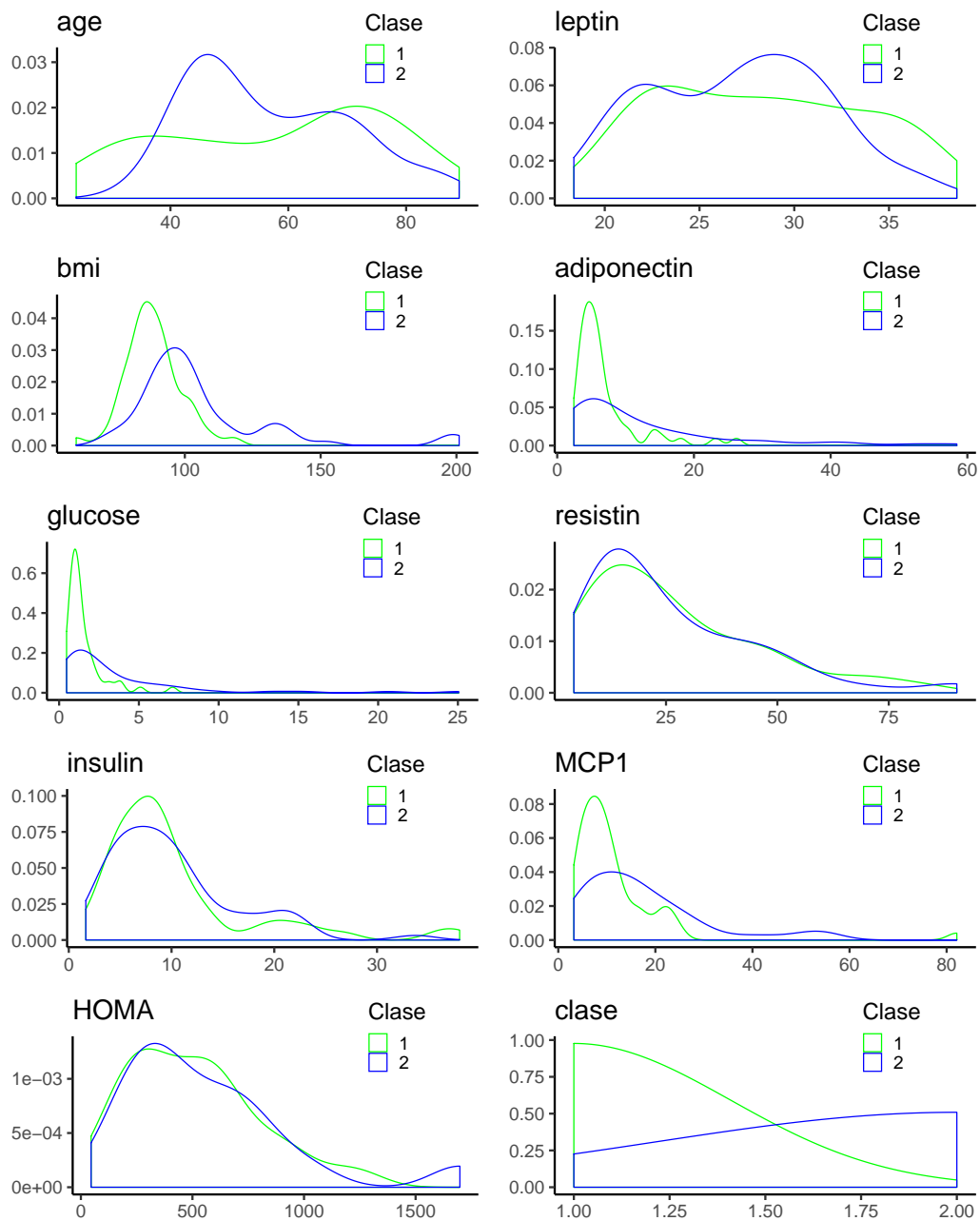


Fig. 4: Gráficos de densidad R.

```

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

Listing 6: Código R generador de los density plots.

4 Boxplot

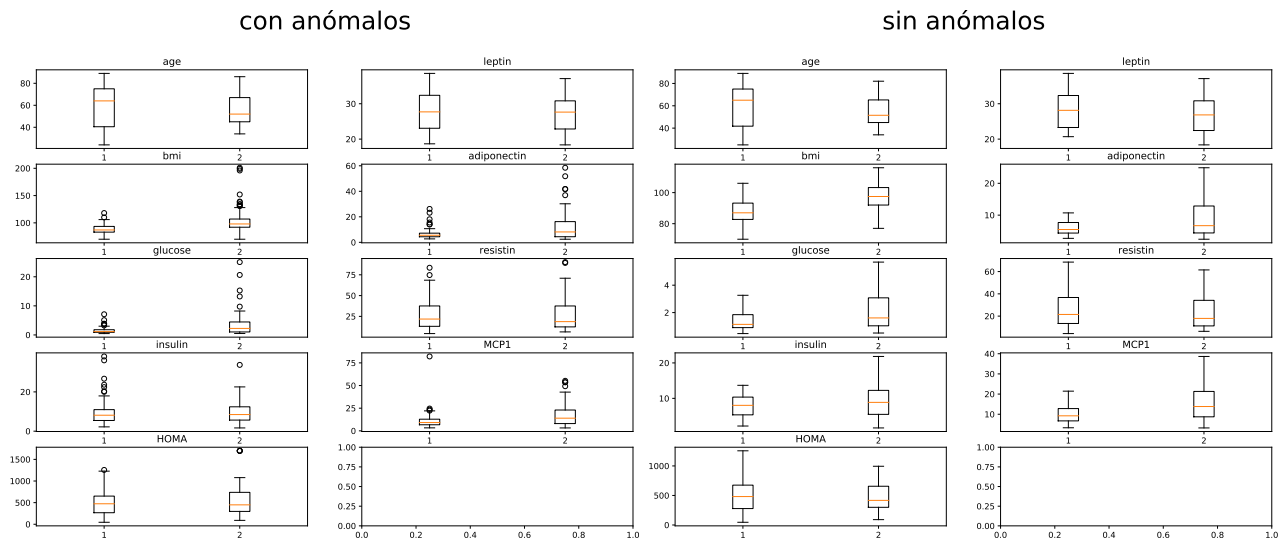


Fig. 5: Boxplots Python para datos con y sin anomalías.

```

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 fig, ax = plt.subplots (nrows = 5, ncols = 2, figsize = (13, 10))
8 ax = ax.flatten ()
9
10 for i in range (0, 9):
11     ax[i].boxplot ([x [y == 1, i], x [y == 2, i]])
12     ax[i].title.set_text (labels [i])
13
14 fig.suptitle ('con anomalos', fontsize = 30)
15 fig.savefig ('../images/boxp.pdf', bbox_inches = 'tight', pad_inches = 0)

```

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

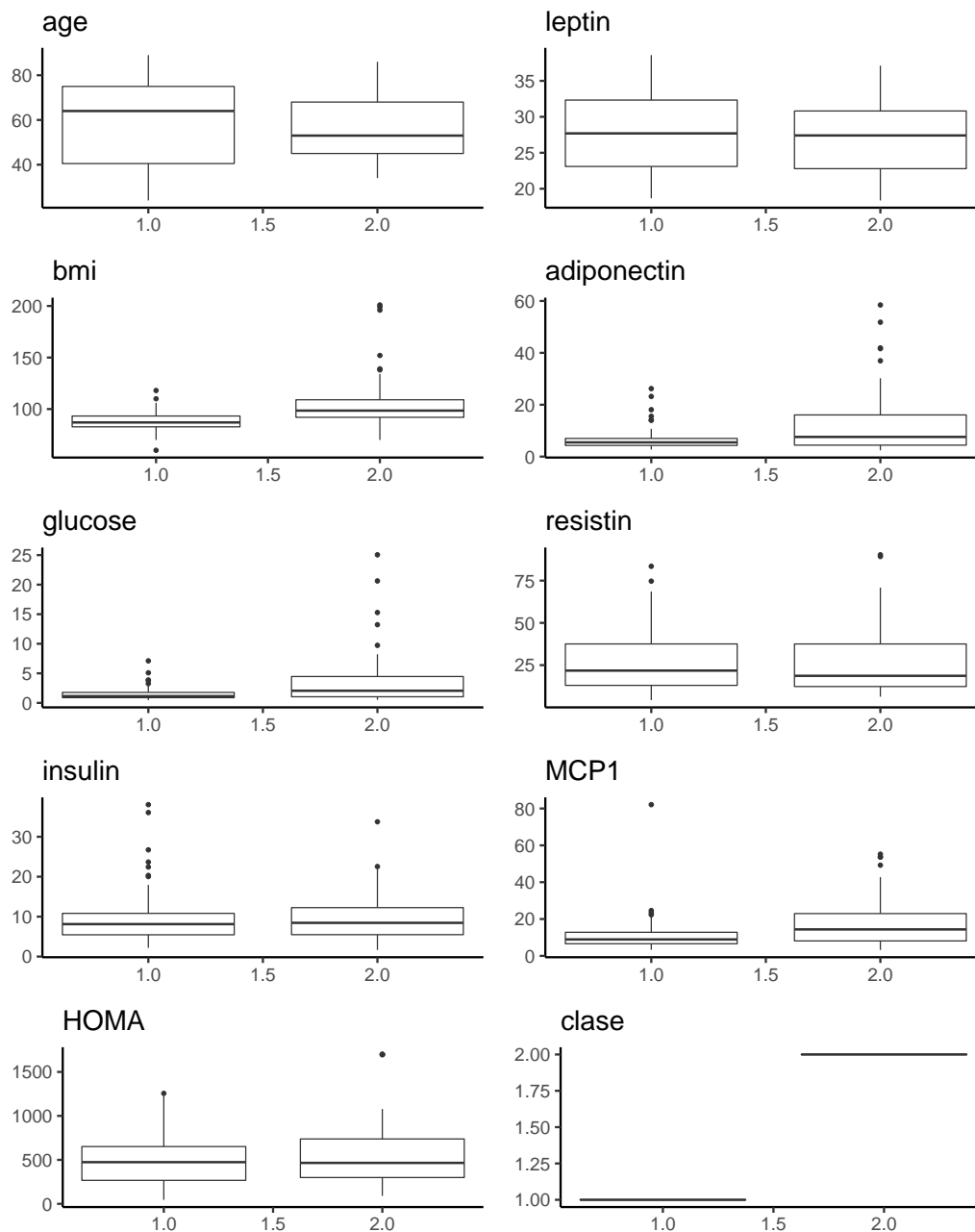


Fig. 6: Boxplots R para datos con anomalías.

```

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

```

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

5 QQplot

con anómalos

sin anómalos

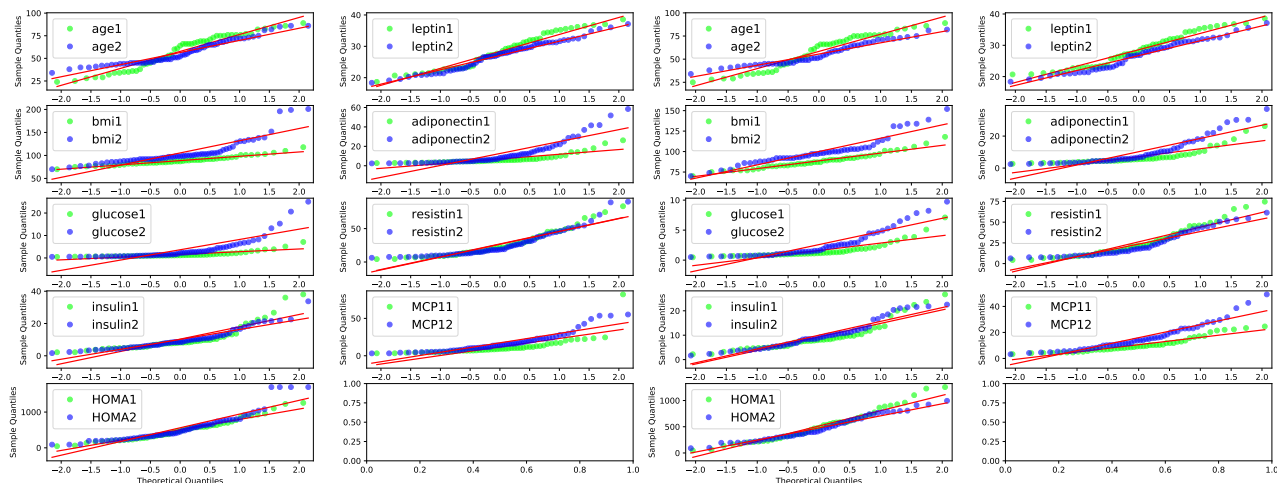


Fig. 7: QQplots Python para datos con y sin anomalías.

```

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  import statsmodels.api as sm
7
8  fc = [(0, (0, 1, 0, 0.6), (0, 0, 1, 0.6))]
9
10 fig, ax = plt.subplots (nrows = 5, ncols = 2, figsize = (13, 10))
11 ax = ax.flatten ()
12
13 for i in range (0, 9):
14     for j in [1, 2]:
15         sm.qqplot (x [y == j, i], ax = ax[i], c = fc[j],
16                     line = 's', label = labels [i] + str (j))
17         ax[i].legend (loc = 2, prop={'size': 15})
18
19 fig.suptitle ('con anómalos', fontsize = 30)
20 fig.savefig ('../images/qqp.pdf', bbox_inches = 'tight', pad_inches = 0)

```

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

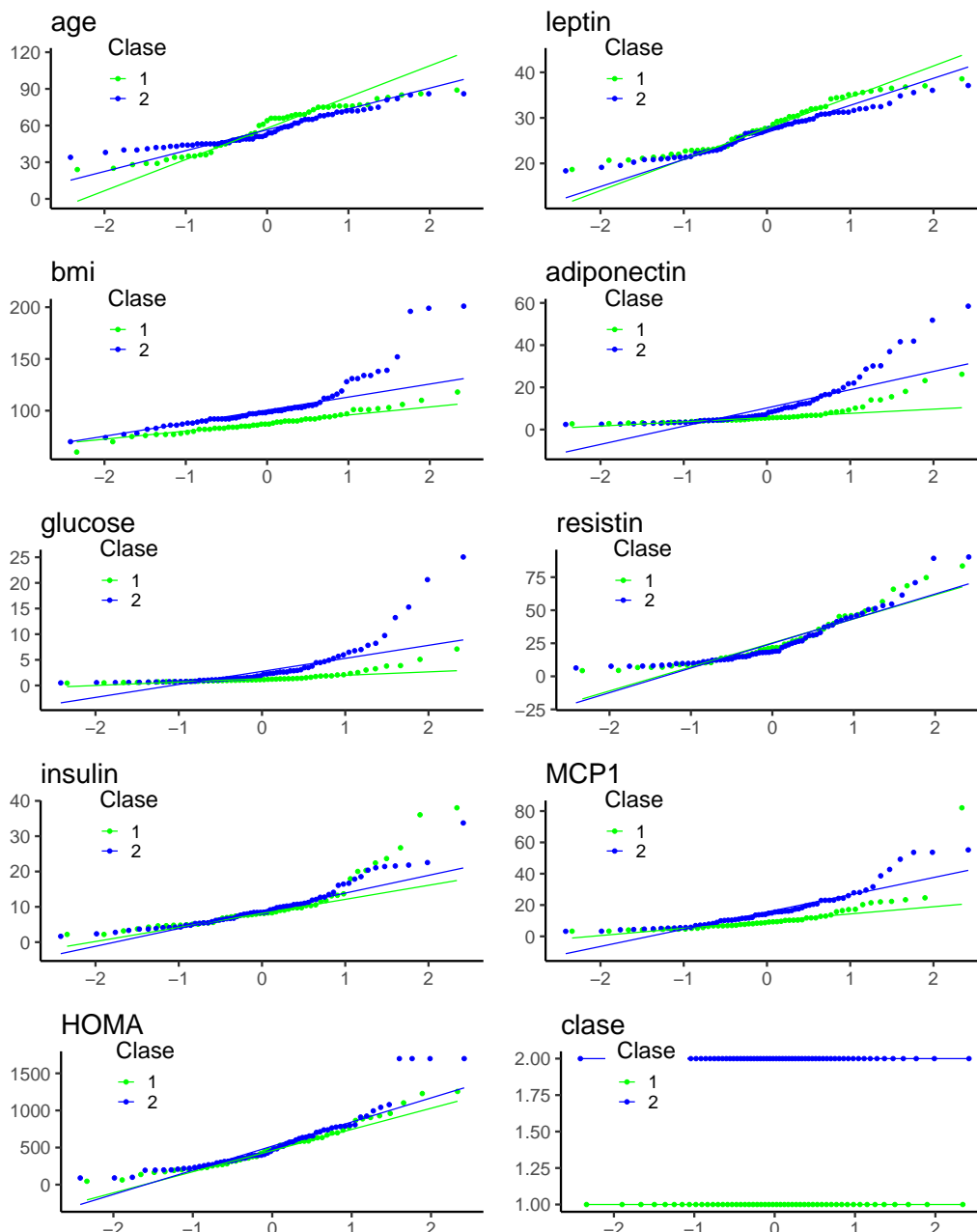


Fig. 8: QQplots R.

```

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

```

Listing 10: Código R generador de los QQplots.

6 Corrplot

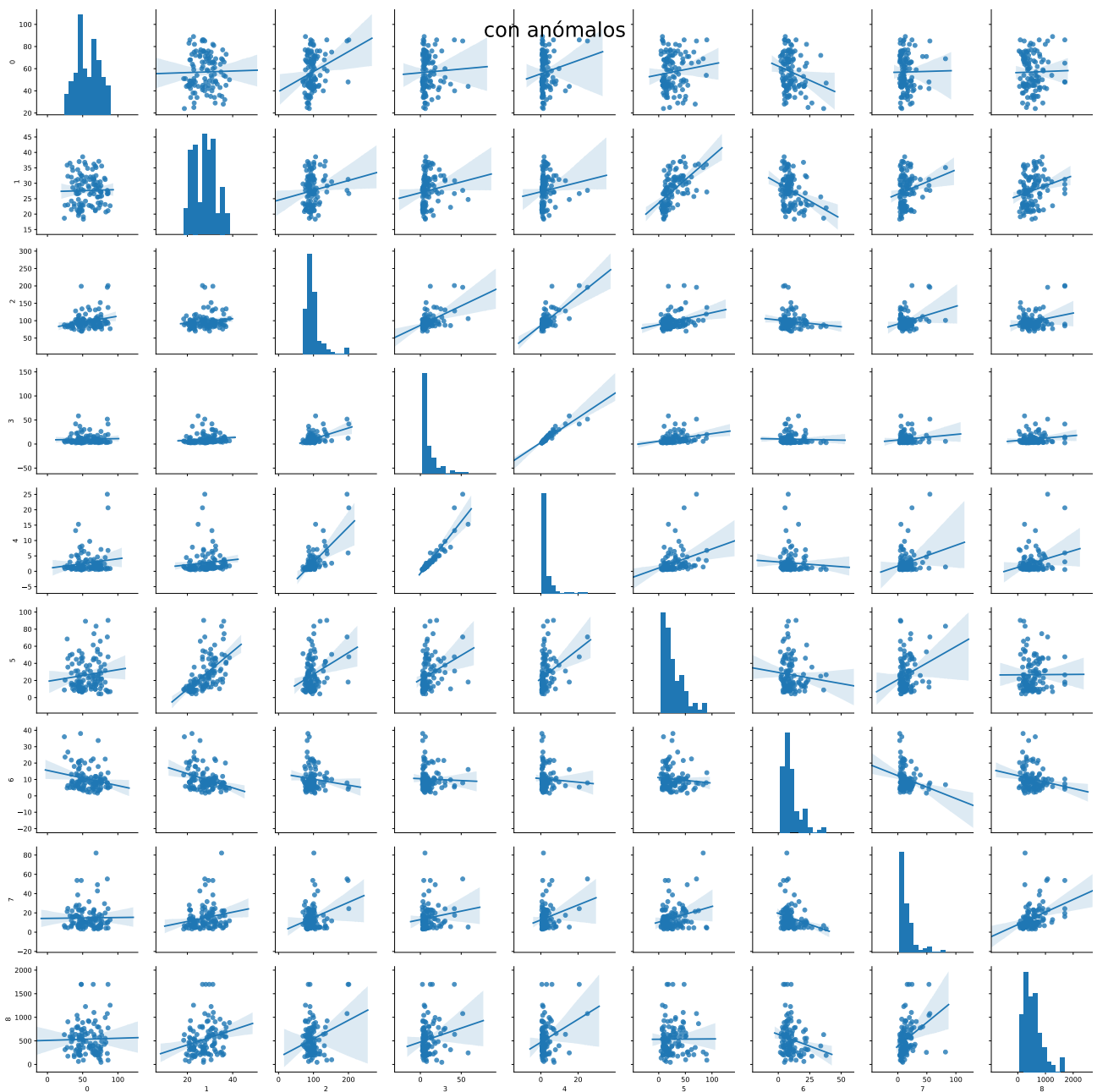


Fig. 9: Corrplot Python para datos con anomalías.

```

1 import pandas as pd
2 import seaborn as sns
3 dataframe = pd.DataFrame.from_records(x)
4 sns.pairplot (dataframe, kind = 'reg')
5 plt.suptitle ('con anomalías', fontsize = 30)
6 plt.savefig ('../images/corrp.pdf', bbox_inches = 'tight', pad_inches = 0)

```

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

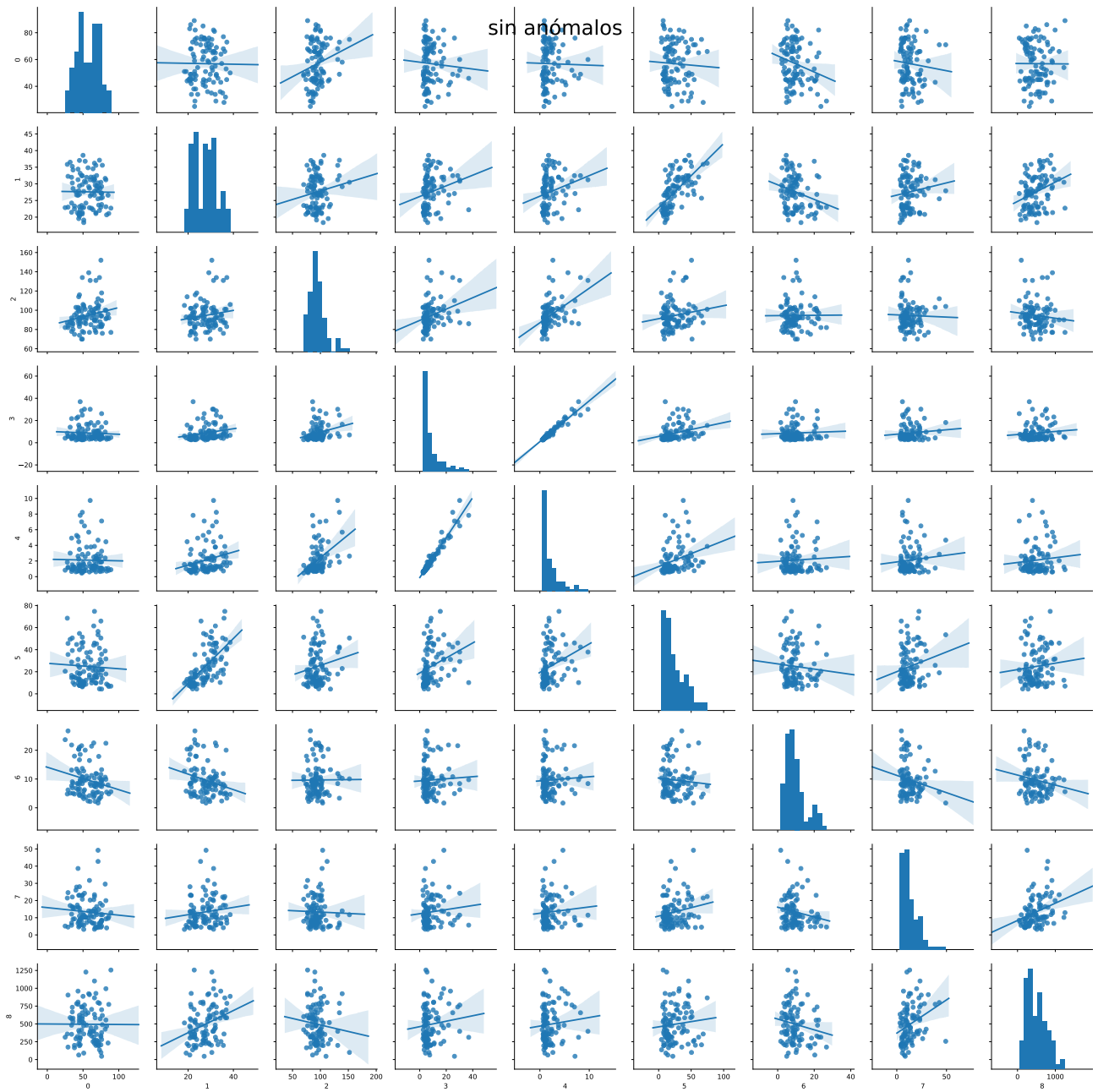


Fig. 10: Corrplot Python para datos sin anomalías.

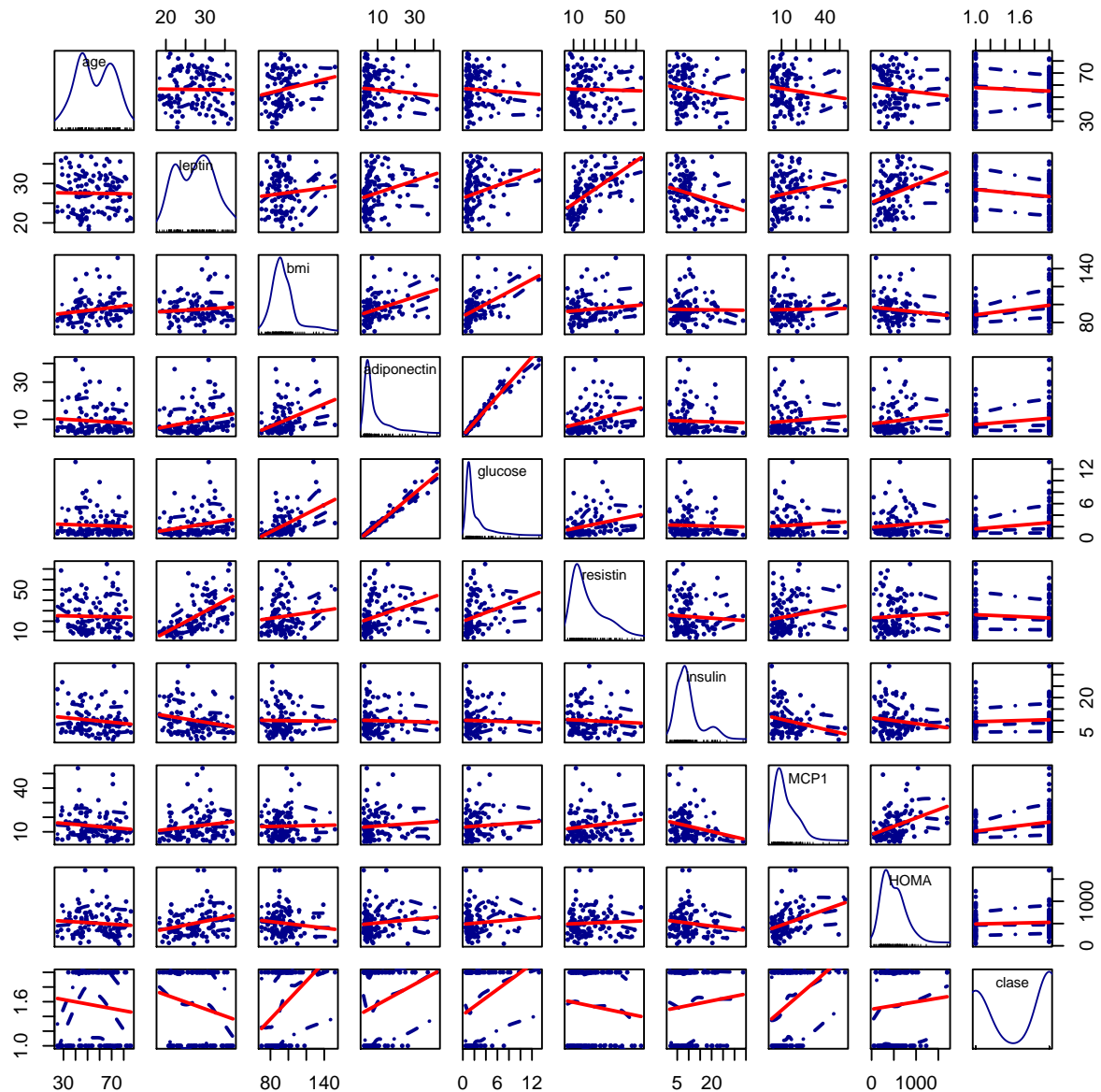


Fig. 11: Corrplot R para datos con anomalías.

```

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 12: Código R generador de los corrplots.

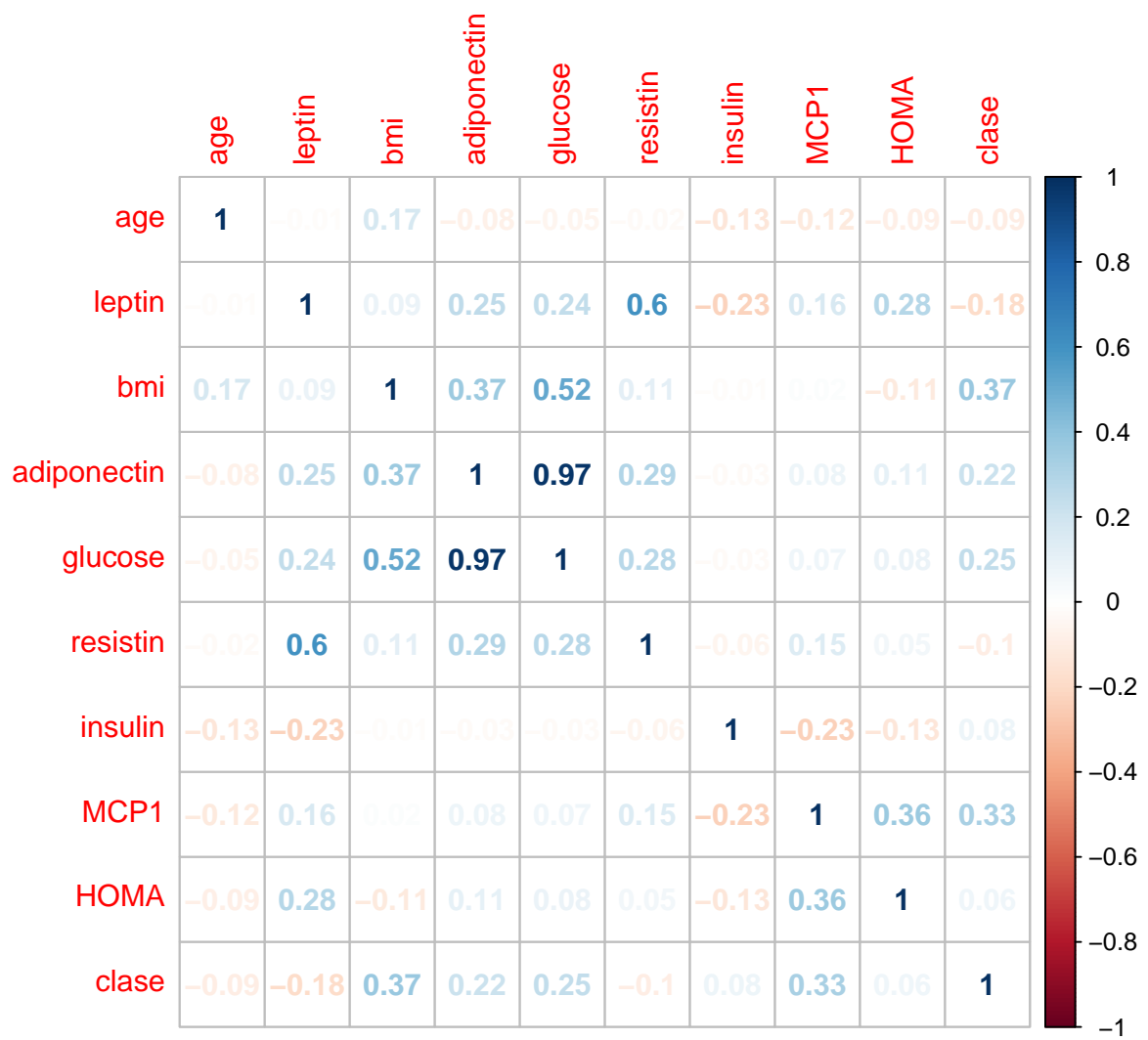


Fig. 12: Matriz de correlaciones en R.

7 Filter Methods

```
1 # Filter Methods
2 import sklearn.feature_selection as sk
3
4 Fscore, pval = sk.f_classif (x_no, y_no)
5 r1 = Fscore.argsort().argsort() # fscore rank
6 print (r1+1)
7
8 import ReliefF as rl
9
10 r2 = rl.ReliefF (n_neighbors = 1) # relieff rank
11 r2.fit(x_no, y_no)
12 r2 = r2.top_features
13 print (r2+1)
14
15 diferencias = abs (r1-r2)
16 media = np.mean (diferencias)
```

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

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

Listing 14: Ranking de variables según los métodos filter.

```

1 # Fscore
2 library (PredPsych)
3 rank (fscore (datos, 10, 1:9))
4 # age leptin bmi adiponectin glucose resistin insulin MCP1 HOMA
5 # 3 5 9 7 8 2 1 6 4
6
7 # Relieff
8 brary (CORElearn)
9 rank (attrEval (as.factor (clase)~., datos, 'Relief'))
10 # age leptin bmi adiponectin glucose resistin insulin MCP1 HOMA
11 # 9 7 8 2 4 5 1 6 3
12
13 # Algunos de los posibles metodos
14 for (i in infoCore (what = "attrEval")){
15     cat (i, '\r\t\t', unname (rank (attrEval (as.factor (clase)~., datos, i))),'\n')
16 }
17 # ReliefFequalK 9 3 8 4 6 5 2 7 1
18 # ReliefFexpRank 8 5 9 3 6 4 1 7 2
19 # ReliefFbestK 9 7 8 3 4 5 1 6 2
20 # Relief 9 7 8 2 4 5 1 6 3
21 # InfGain 7 4 9 5 8 2 1 6 3
22 # GainRatio 9 2 8 7 6 4.5 1 3 4.5
23 # MDL 7 4 9 5 8 3 1 6 2
24 # Gini 7 4 9 5 8 3 1 6 2
25 # MyopicReliefF 6 4 9 5 7 3 1 8 2
26 # Accuracy 6 4 9 5 7 3 1.5 8 1.5
27 # ReliefFmerit 8 3 9 5 6 4 1 7 2
28 # ReliefFdistance 8 4 9 5 6 3 1 7 2
29 # ReliefFsqrDistan 8 4 9 5 6 3 1 7 2
30 # DKM 7 3 9 6 8 2 1 5 4
31 # ReliefFexpC 8 5 9 3 6 4 1 7 2
32 # ReliefFavgC 8 5 9 3 6 4 1 7 2
33 # ReliefFpe 8 5 9 3 6 4 1 7 2
34 # ReliefFpa 8 5 9 3 6 4 1 7 2
35 # ReliefFsmp 8 5 9 3 6 4 1 7 2
36 # GainRatioCost 9 2 8 7 6 4.5 1 3 4.5
37 # DKMcost 7 4 9 5 8 3 2 6 1
38 #

```

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

8 Wrapper Methods

```
1 from sklearn.neighbors import KNeighborsClassifier
2 from mlxtend.feature_selection import SequentialFeatureSelector
3
4 knn = KNeighborsClassifier (n_neighbors = 50)
5
6 sfs = SequentialFeatureSelector (knn,
7                                 k_features = 4,
8                                 forward = True,
9                                 scoring = 'accuracy',
10                                cv = 10)
11
12 sfs.fit (x_no, y_no, custom_feature_names = labels)
13 print (sfs.k_score_)
14 print ('Sequential Forward Selection', sfs.k_feature_names_, end = '\n\n')
15
16 sfs.forward = False
17
18 sfs.fit (x_no, y_no, custom_feature_names = labels)
19 print (sfs.k_score_)
20 print ('Sequential Backward Selection', sfs.k_feature_names_, end = '\n\n')
```

Listing 16: Aplicación métodos *wrapper* de selección características.

```
1 0.7054545454545454
2 Sequential Forward Selection ('leptin', 'bmi', 'glucose', 'MCP1')
3
4 0.7094949494949495
5 Sequential Backward Selection ('leptin', 'bmi', 'glucose', 'insulin')
```

Listing 17: Resultados Python del filtrado mediante wrappers.

```

1 # Sequential Feature Selector
2 library (mlr)
3 # Forward
4 sfs <- selectFeatures (
5     learner      = makeLearner      ('classif.knn', k = 9, l = 3),
6     task         = makeClassifTask  (data = datos, target = 'clase'),
7     resampling    = makeResampleDesc ("CV", iter = 50),
8     control       = makeFeatSelControlSequential (method = "sfs", maxit = 100L))
9 # FeatSel result:
10 # Features (4): age, leptin , bmi, MCP1
11 # mmce.test.mean=0.1833333
12
13 # Backward
14 sbs <- selectFeatures (
15     learner      = makeLearner      ('classif.knn', k = 9, l = 3),
16     task         = makeClassifTask  (data = datos, target = 'clase'),
17     resampling    = makeResampleDesc ("CV", iter = 50),
18     control       = makeFeatSelControlSequential (method = "sbs", maxit = 100L))
19 # FeatSel result:
20 # Features (4): age, leptin , bmi, MCP1
21 # mmce.test.mean=0.1800000

```

Listing 18: Resultados R del filtrado mediante wrappers.

```

1 # esto es extra
2 library (Boruta)
3 Boruta (as.factor (clase)~., datos, maxRuns = 101) -> borutaout
4
5 # Boruta performed 100 iterations in 4.317041 secs.
6 # 5 attributes confirmed important: age, bmi, glucose , leptin , MCP1;
7 # 3 attributes confirmed unimportant: HOMA, insulin , resistin;
8 # 1 tentative attributes left: adiponectin;
9
10 pdf ("../images/boruta.pdf")
11 plot (borutaout, las = 2, xlab = '', main = 'Boruta Variable Importance')
12 dev.off ()

```

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

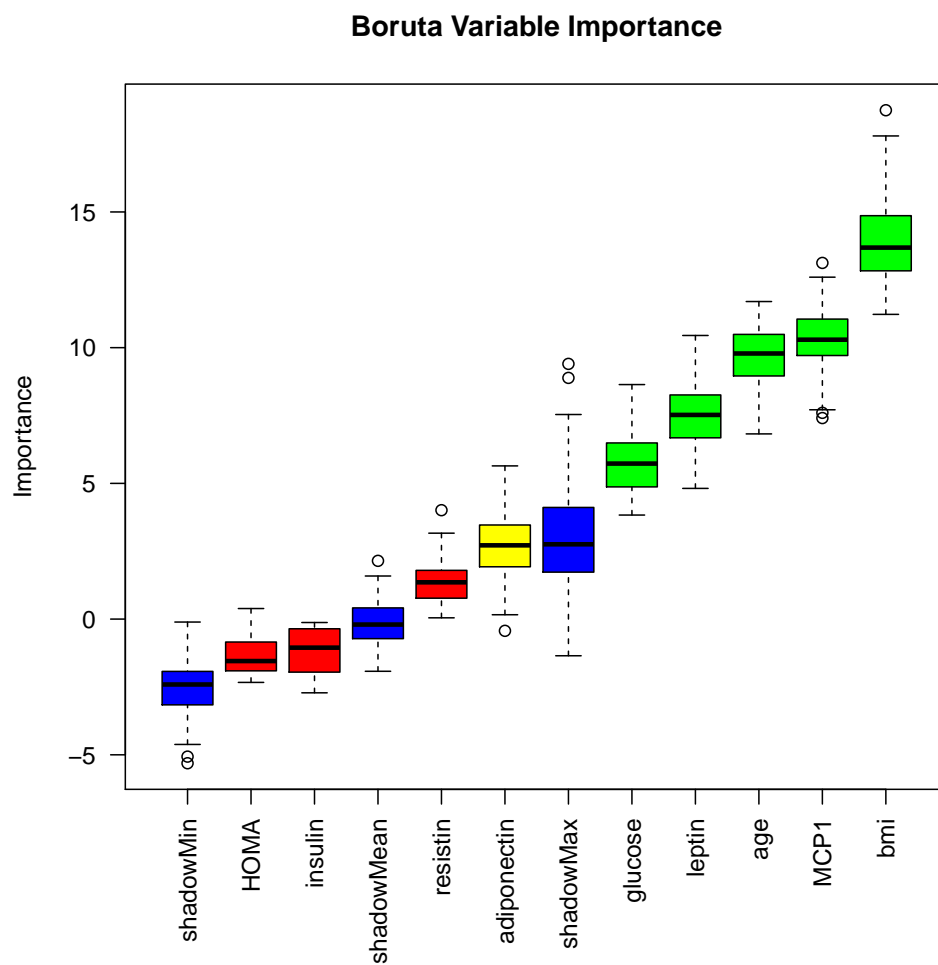


Fig. 13: Representación gráfica de la importancia de las variables seleccionadas por Boruta.

9 PCA

```
1 from sklearn.preprocessing import StandardScaler
2 x_no = StandardScaler ().fit_transform (x_no) # typify
3 from sklearn.decomposition import PCA
4 pca = PCA (n_components = 9)
5 principalComponents = pca.fit_transform(x_no)
6 evr = pca.explained_variance_ratio_
```

Listing 20: *Principal Component Analysis* Python.

```
1 [0.29146865 0.18490568 0.14125105 0.11727276 0.08486126 0.07999359
2 0.06636991 0.03254865 0.00132847]
3 [0.29146865 0.47637432 0.61762537 0.73489813 0.81975939 0.89975298
4 0.96612289 0.99867153 1.          ]
```

Listing 21: Varianza explicada por componente y suma acumulada Python.

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

Listing 22: *Principal Component Analysis* R.

```
1 Importance of components:
2          PC1    PC2    PC3    PC4    PC5    PC6    PC7    PC8    PC9
3 Std deviation  1.7475 1.2393 1.082 1.048 0.8528 0.8144 0.66261 0.53101 0.17555
4 Propor. of Var. 0.3393 0.1707 0.130 0.122 0.0808 0.0737 0.04878 0.03133 0.00342
5 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

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

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

```

1 pdf ("../images/pareto.pdf", width = 7, height = 5.5)
2 x <- pca[['sdev']]^2
3 cx <- cumsum (x)
4 par (mar = c(3,3,4,3))
5 pc <- barplot (x, names.arg = dimnames (pca[['rotation']])[[2]],
6               border = NA, axes = F, main = 'Pareto R',
7               ylim = c(0, 1.05*max(cx, na.rm = T)), col = 'blue4'
8 )
9 lines (pc, cx, type = 'b', pch = 19, col="red")
10 box (col      = 'black')
11 axis (side    = 2,
12       at      = c (0, round (x[c (1,2,4,6,8,9)], 1)),
13       las     = 2, cex.axis = 0.8,
14 )
15 axis (side    = 4,
16       at      = c(0, cx[1:8]),
17       labels   = paste (c (0, round (cx[1:8]/max (cx) * 100)) , "%", sep=""),
18       las     = 2, cex.axis = 0.8
19 )
20 dev.off ()

```

Listing 25: Código generador del diagrama de Pareto en R.

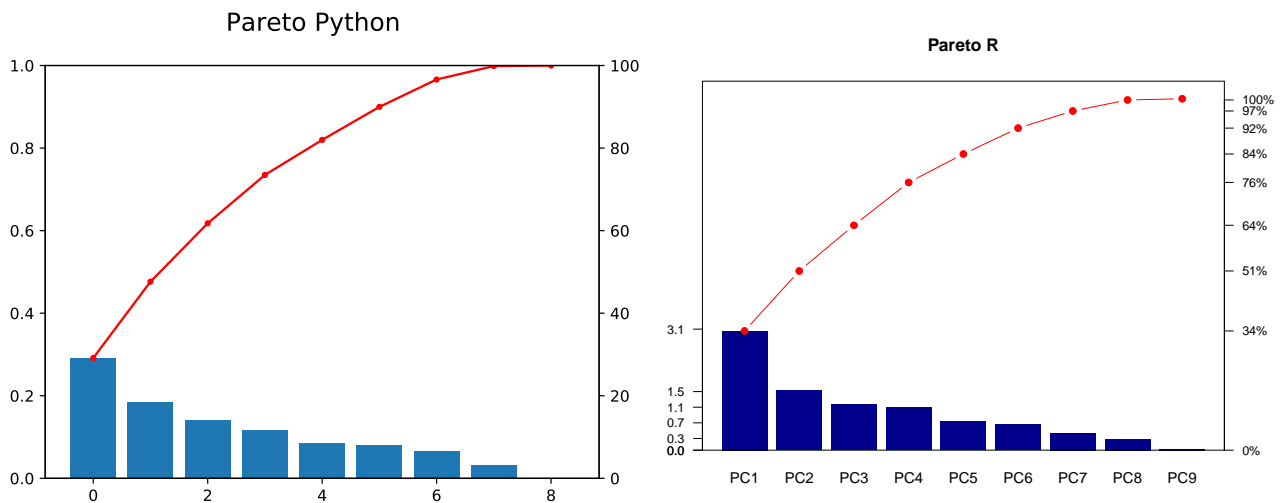


Fig. 14: Diagrama de Pareto en Python y R.

9.2 Biplot

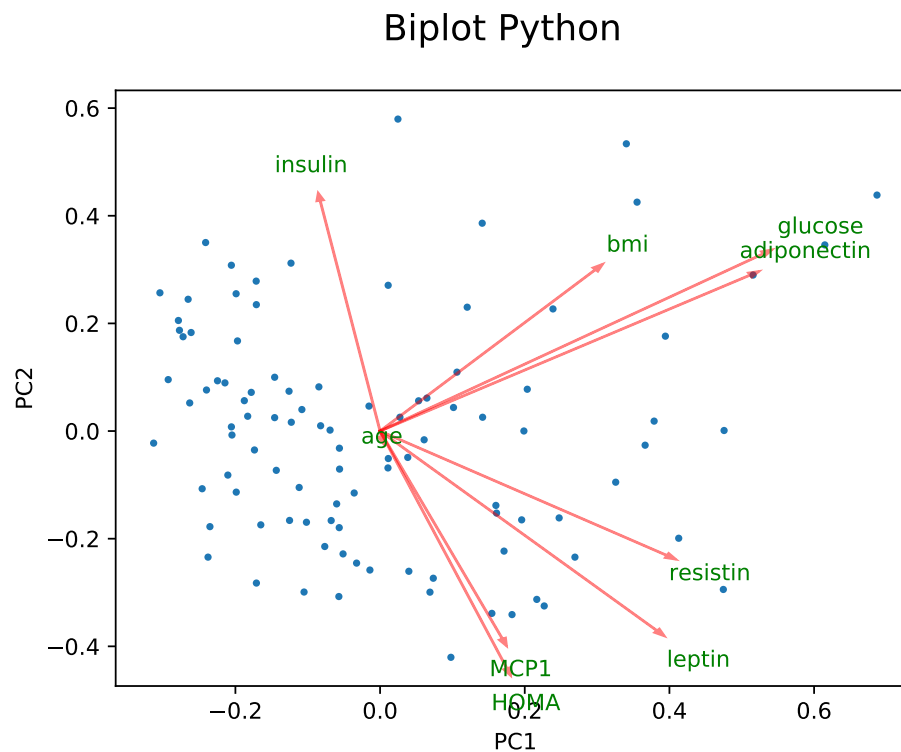


Fig. 15: Biplot Python.

```
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 = 'g',
13                      ha = 'center', va = 'center')
14         else:
15             plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, labels[i], color = 'g', ha =
16                      = 'center', va = 'center')
17     plt.xlabel("PC{}".format(1)), plt.ylabel("PC{}".format(2))
18     plt.savefig ('../images/biplotpca.pdf', bbox_inches = 'tight', pad_inches = 0)
19 biplot (principalComponents[:,0:2], np.transpose(pca.components_[0:2, :]), labels)
```

Listing 26: Código generador del Biplot en Python.

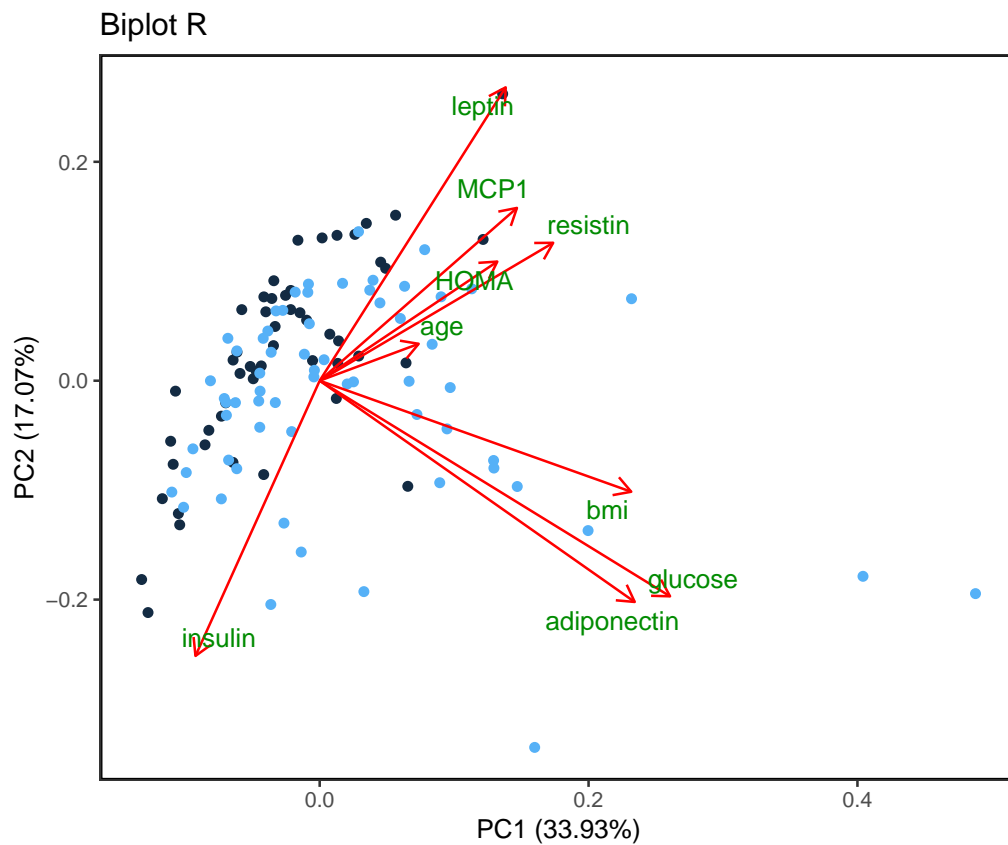


Fig. 16: Biplot R.

```

1 library (ggfortify)
2 pdf ("../images/biplot.pdf", width = 6, height = 5)
3 autoplot (pca, data = datos, colour = 'clase',
4           loadings = T,
5           main = 'Biplot R',
6           loadings.label = T,
7           loadings.label.repel = T,
8           loadings.label.colour = 'green4',
9 ) +
10 theme_bw () +
11 theme (panel.grid.major = element_blank(),
12       panel.grid.minor = element_blank(),
13       panel.background = element_rect(colour = "black", size = 1),
14       legend.position = 'none'
15 )

```

Listing 27: Código generador del Biplot en R.

10 Modelos de Clasificación

10.1 Clasificación Lineal

```
1 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
2
3 lda = LDA ()
4 score = cross_val_score (lda, x, y, cv = 10)
5 print ('Linear puntuación CV media: %.2f std: %.2f'
6       %(np.mean (score), np.std (score)))
7
8 score = cross_val_score (lda, x, y, cv = KFold (n_splits = 10, shuffle = True))
9 print ('Linear puntuación KF media: %.2f std: %.2f'
10      %(np.mean (score), np.std (score)))
11
12 score = cross_val_score (lda, x, y, cv = ShuffleSplit (n_splits = 10))
13 print ('Linear puntuación SS media: %.2f std: %.2f'
14      %(np.mean (score), np.std (score)))
15
16 score = cross_val_score (lda, x, y, cv = LeaveOneOut ())
17 print ('Linear puntuación LO media: %.2f std: %.2f'
18      %(np.mean (score), np.std (score)))
```

Listing 28: Python validación del modelo lineal.

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

Listing 29: Python validación según distintos métodos de partición.

```

1 # Linear Discriminant Analysis
2 it <- 1000
3 ldascores <- rep (NA, times = it)
4 p <- 0.7 # partition
5 cat ('LDA\n')
6 pb <- txtProgressBar (min = 0, max = it, initial = 0, char = '|', style = 3)
7 for (i in 1:it){
8   train.samples <- datos$clase %>% createDataPartition (p = p, list = F)
9
10  train.data      <- datos[ train.samples,]
11  test.data       <- datos[-train.samples,]
12
13  preproc.param <- train.data %>% preProcess (method = c ("center", "scale"))
14
15  train.trans    <- preproc.param %>% predict (train.data)
16  test.trans     <- preproc.param %>% predict (test.data)
17
18  mdl <- lda (clase~., data = train.trans)
19
20  prd <- mdl %>% predict (test.trans)
21
22  ldascores[i]   <- mean (prd$class == test.trans$clase)
23  setTxtProgressBar (pb, i)
24 }

```

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

```
1 from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
2
3 qda = QDA ()
4 score = cross_val_score (qda, x, y, cv = 10)
5 print ('Quadratic puntuación CV media: %.2f std: %.2f'
6       %(np.mean (score), np.std (score)))
7
8 score = cross_val_score (qda, x, y, cv = KFold (n_splits = 10, shuffle = True))
9 print ('Quadratic puntuación KF media: %.2f std: %.2f'
10      %(np.mean (score), np.std (score)))
11
12 score = cross_val_score (qda, x, y, cv = ShuffleSplit (n_splits = 10))
13 print ('Quadratic puntuación SS media: %.2f std: %.2f'
14      %(np.mean (score), np.std (score)))
15
16 score = cross_val_score (qda, x, y, cv = LeaveOneOut ())
17 print ('Quadratic puntuación LO media: %.2f std: %.2f'
18      %(np.mean (score), np.std (score)))
```

Listing 32: Python validación del modelo cuadrático.

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

Listing 33: Python validación según distintos métodos de partición.

```

1 qdascores <- rep (NA, times = it)
2 cat ('\nQDA\n')
3 pb <- txtProgressBar (min = 0, max = it, initial = 0, char = '|', style = 3)
4 for (i in 1:it){
5   train.samples <- datos$clase %>% createDataPartition (p = p, list = F)
6
7   train.data      <- datos[ train.samples,]
8   test.data       <- datos[-train.samples,]
9
10  preproc.param <- train.data %>% preProcess (method = c ("center", "scale"))
11
12  train.trans    <- preproc.param %>% predict (train.data)
13  test.trans     <- preproc.param %>% predict (test.data)
14
15  mdl <- qda (clase~., data = train.trans)
16
17  prd <- mdl %>% predict (test.trans)
18
19  qdascores[i]   <- mean (prd$class == test.trans$clase)
20
21  setTxtProgressBar (pb, i)
22 }

```

Listing 34: R análisis cuadrático discriminante.

| | vars | n | mean | sd | median | trimmed | mad | min | max | range | skew |
|---|-----------|--------|------|------|--------|---------|------|------|------|-------|-------|
| 1 | | | | | | | | | | | |
| 2 | 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

```
1 from sklearn.neighbors import KNeighborsClassifier
2
3 knn = KNeighborsClassifier (n_neighbors = 9)
4 score = cross_val_score (knn, x, y, cv = 10)
5 print ('KNN puntuación CV media: %.2f std: %.2f'
6         %(np.mean (score), np.std (score)))
7
8 score = cross_val_score (knn, x, y, cv = KFold (n_splits = 10, shuffle = True))
9 print ('KNN puntuación KF media: %.2f std: %.2f'
10        %(np.mean (score), np.std (score)))
11
12 score = cross_val_score (knn, x, y, cv = ShuffleSplit (n_splits = 10))
13 print ('KNN puntuación SS media: %.2f std: %.2f'
14        %(np.mean (score), np.std (score)))
15
16 score = cross_val_score (knn, x, y, cv = LeaveOneOut ())
17 print ('KNN puntuación LO media: %.2f std: %.2f'
18        %(np.mean (score), np.std (score)))
```

Listing 36: Python validación del modelo KNN.

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

Listing 37: Python validación según distintos métodos de partición.

```

1 knnscores <- rep (NA, times = it)
2 library (class)
3 cat ('\nKNN\n')
4 pb <- txtProgressBar (min = 0, max = it, initial = 0, char = '|', style = 3)
5 for (i in 1:it){
6   train.samples <- datos$clase %>% createDataPartition (p = p, list = F)
7
8   train.data <- datos[ train.samples,]
9   test.data <- datos[-train.samples,]
10
11   preproc.param <- train.data %>% preProcess (method = c ("center", "scale"))
12
13   train.trans <- preproc.param %>% predict (train.data)
14   test.trans <- preproc.param %>% predict (test.data)
15
16   prd <- knn (train = train.trans[1:9],
17             cl = train.trans$clase,
18             test = test.trans[1:9],
19             k = 1)
20
21   knnscores[i] <- mean (prd == test.trans$clase)
22
23   setTxtProgressBar (pb, i)
24 }

```

Listing 38: R *K nearest neighbours*.

| | vars | n | mean | sd | median | trimmed | mad | min | max | range | skew |
|---|-----------|---|------|------|--------|---------|------|------|------|-------|------------|
| 1 | | | | | | | | | | | |
| 2 | 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

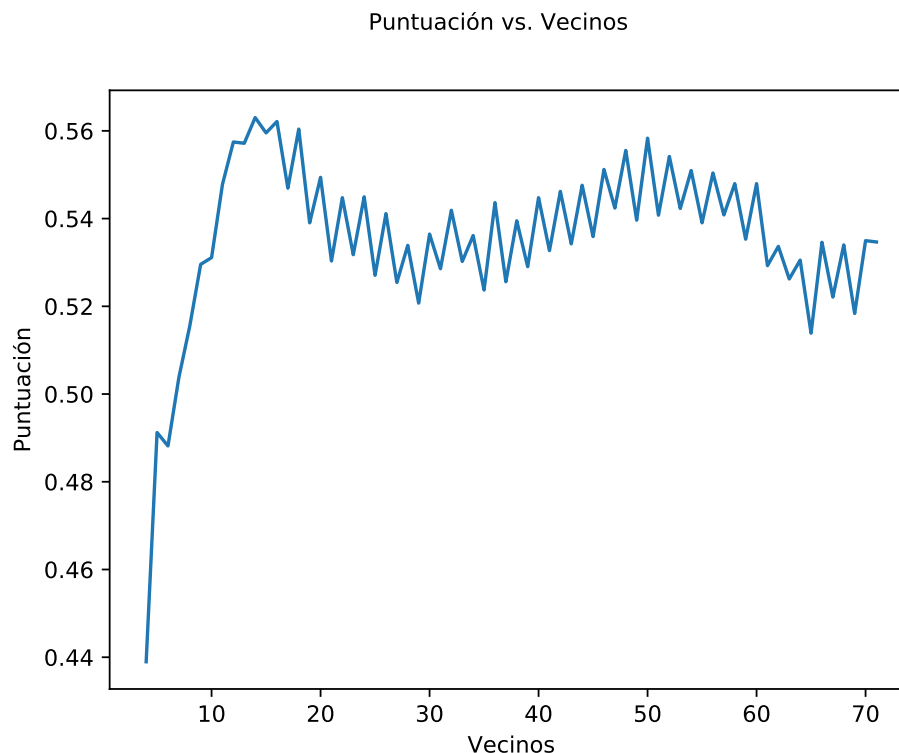


Fig. 17: Rendimiento decreciente según aumenta el número de vecinos.

```
1 score = [None]*(vecinos)
2 for i in range (2,vecinos):
3     print ('n_neighbors = %i'% (i), end = '\r')
4     iteraciones = 1000
5     error = [None]*iteraciones
6     for j in range (0, iteraciones):
7         X_train, X_test, y_train, y_test = train_test_split (x, y, test_size = 0.3)
8         knn = KNeighborsClassifier (n_neighbors = i, n_jobs = -1)
9         knn.fit (X_train, y_train)
10        error[j] = np.sum (abs (knn.predict (X_test) - y_test))/ len (y_test)
11    score[i] = np.mean (error)
12
13 plt.plot (range (2, vecinos+2), score)
14
15 plt.suptitle ('Puntuación vs. Vecinos', fontsize = 10)
16 plt.suptitle ('puntuación vs. vecinos', fontsize = 10)
17 plt.xlabel ('vecinos')
18 plt.ylabel ('puntuación')
19 plt.show ()
```

Listing 40: Evolución de puntuación según número de vecinos.

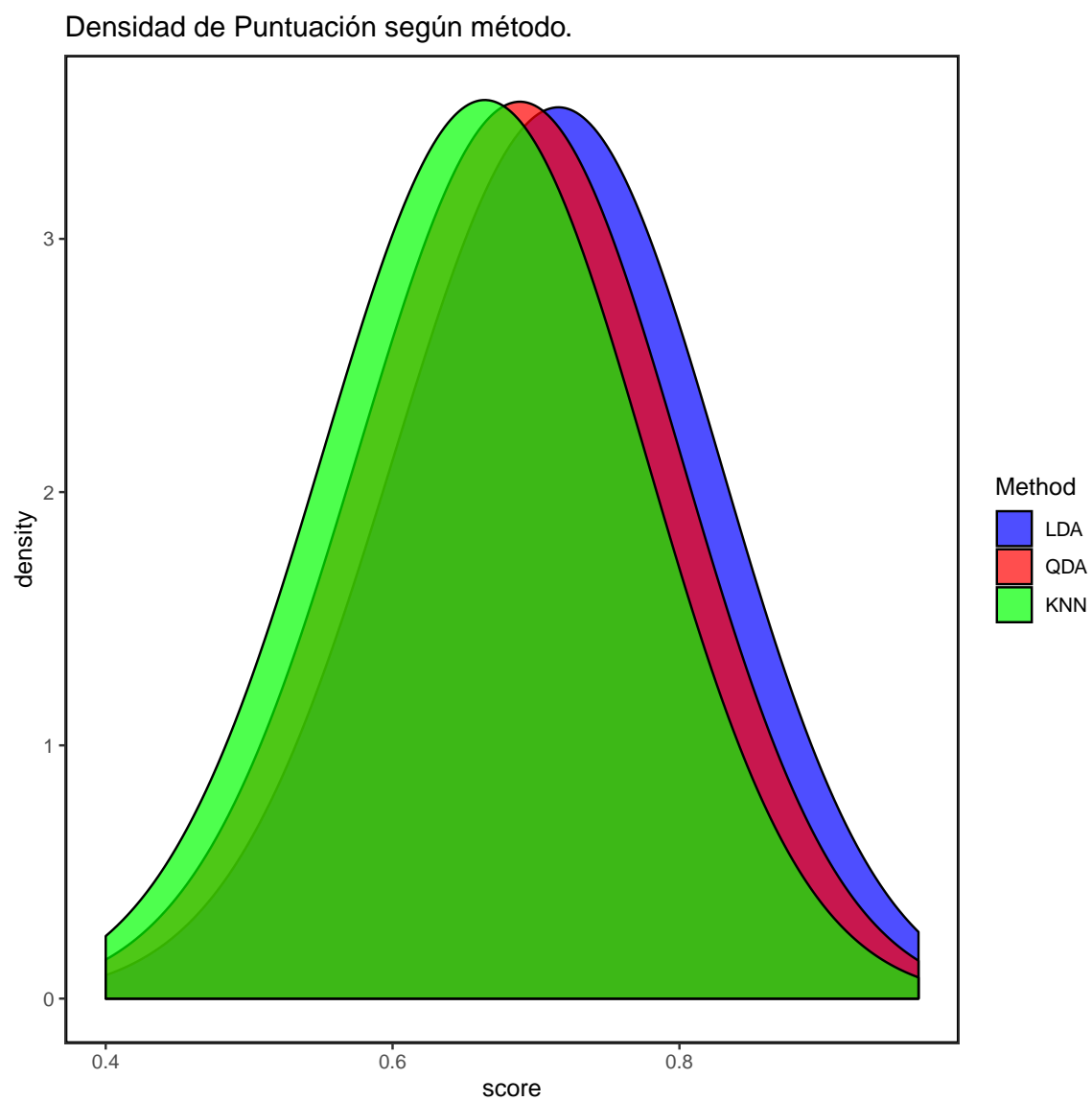


Fig. 18: Comparación de mil evaluaciones de cada uno de los métodos de clasificación en R.