

Sistemas de Información y Telemedicina. *

Marta Girones Sanguesa

Silvia Marset Gomis

Ignacio Amat Hernández

Sofía Gutiérrez Santamaría

December 30, 2019

Contents

| Sección | Página |
|--|-----------|
| 1 Preámbulo | 4 |
| 2 Histogramas | 5 |
| 3 Kernel Density | 7 |
| 4 Boxplot | 8 |
| 5 QQplot | 10 |
| 6 Corrplot | 11 |
| 7 Filter Methods | 13 |
| 8 Wrapper Methods | 14 |
| 9 PCA | 15 |
| 9.1 Pareto | 15 |
| 9.2 Biplot | 16 |
| 10 Modelos de Clasificación | 17 |
| 10.1 Clasificación Lineal | 17 |
| 10.2 Clasificación Cuadrática | 18 |
| 10.3 Clasificación KNN | 19 |

*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 | Boxplots Python para datos con y sin anomalías. | 8 |
| 5 | Boxplots R para datos con anomalías. | 9 |
| 6 | QQplots para datos con y sin anomalías. | 10 |
| 7 | Corrplot para datos con anomalías. | 11 |
| 8 | Corrplot para datos sin anomalías. | 12 |
| 9 | Diagrama de Pareto. | 15 |
| 10 | Biplot. | 16 |
| 11 | Rendimineto decreciente según aumenta el número de vecinos. | 20 |

Listings

| | | |
|----|---|----|
| 1 | Importaciones iniciales y preparacion de datos. | 4 |
| 2 | Código Python generador de los histogramas con datos anómalos. | 5 |
| 3 | Código R generador de los histogramas con datos anómalos. | 6 |
| 4 | Código Python generador de los kernel density plots con datos anómalos. | 7 |
| 5 | Código Python generador de los boxplots con datos anómalos. | 8 |
| 6 | Código R generador de los boxplots con datos anómalos. | 9 |
| 7 | Código generador de los QQplots con datos anómalos. | 10 |
| 8 | Código generador de los corrplots con datos anómalos. | 11 |
| 9 | Aplicación métodos <i>filter</i> de selección características. | 13 |
| 10 | Ranking de variables según los métodos filter. | 13 |
| 11 | Aplicación métodos <i>wrapper</i> de selección características. | 14 |
| 12 | Resultados del filtrado mediante wrappers. | 14 |
| 13 | <i>Principal Component Analysis</i> | 15 |
| 14 | Varianza explicada por componente y suma acumulada. | 15 |
| 15 | Código generador del diagrama de Pareto | 15 |
| 16 | Código generador del Biplot. | 16 |
| 17 | Validación del modelo lineal. | 17 |
| 18 | Validación según distintos métodos. | 17 |
| 19 | Validación del modelo cuadrático. | 18 |
| 20 | Validación según distintos métodos. | 18 |
| 21 | Validación del modelo KNN. | 19 |
| 22 | Validación según distintos métodos. | 19 |
| 23 | Evolución de puntuación según número de vecinos. | 20 |

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.

2 Histogramas

En este apartado dibujamos los histogramas comparativos.

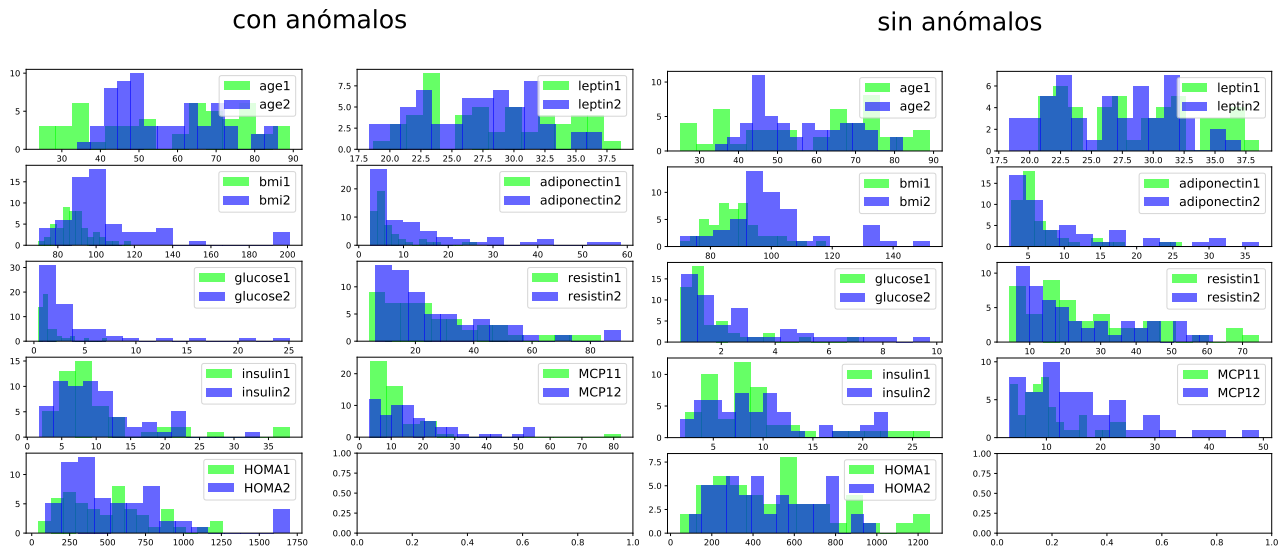


Fig. 1: Histogramas Python para datos con y sin anomalías.

```

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,  $\alpha$ ) ← 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 2: Código Python generador de los histogramas con datos anómalos.

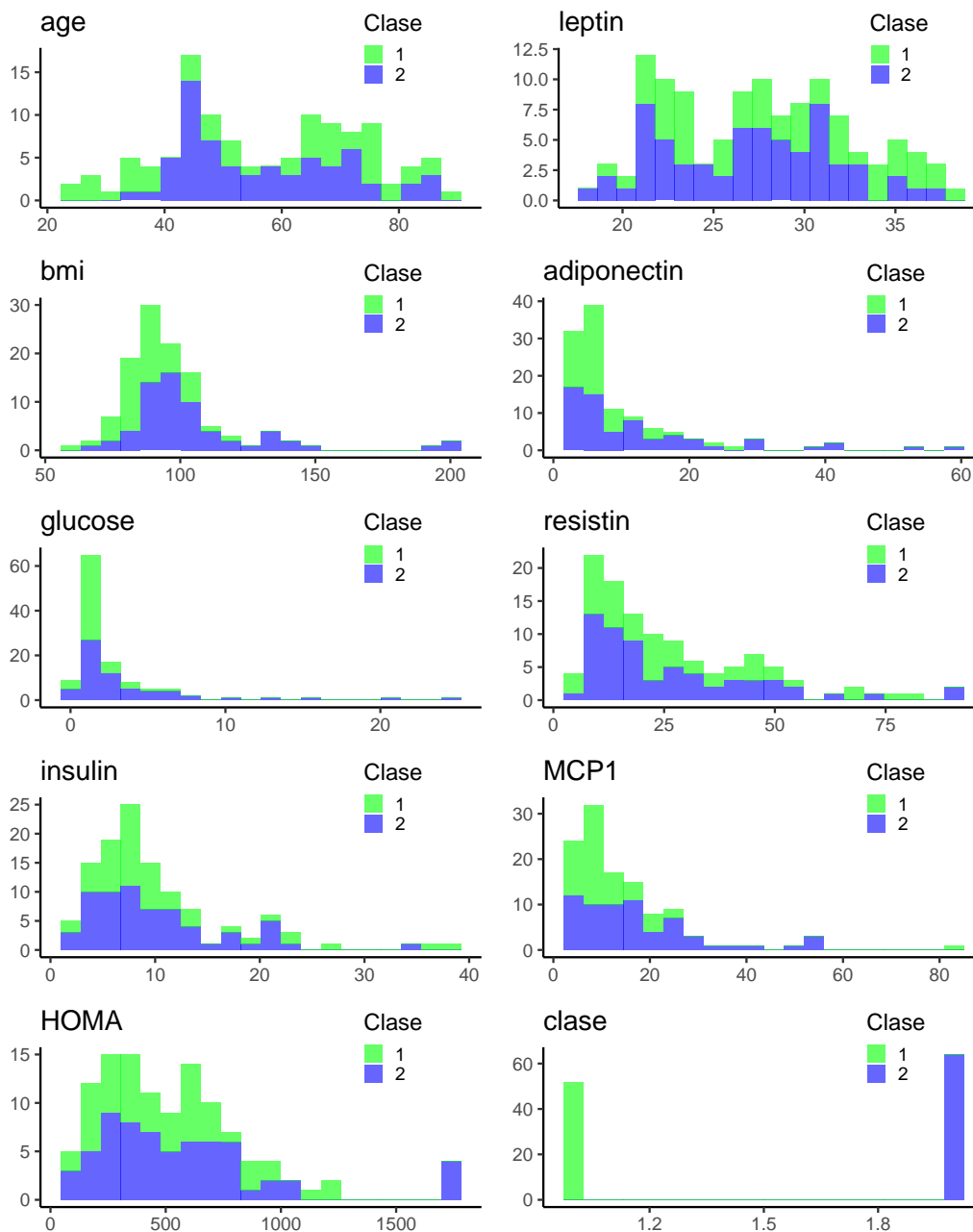


Fig. 2: Histogramas R para datos con anomalías.

```

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 3: Código R generador de los histogramas con datos anómalos.

3 Kernel Density

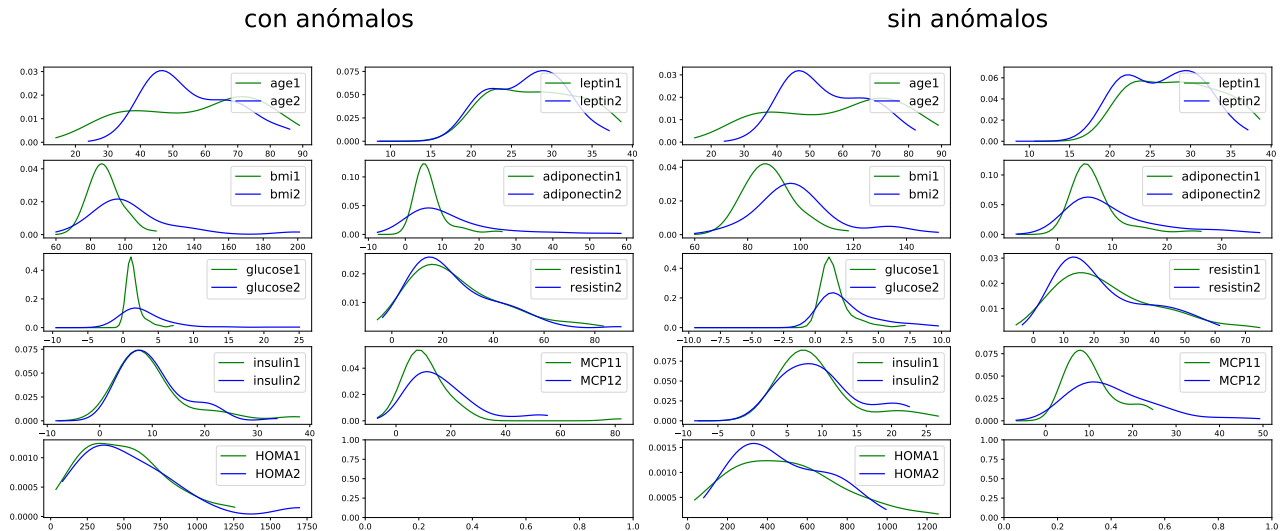


Fig. 3: Kernel Density para datos con y sin anomalías.

```

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

4 Boxplot

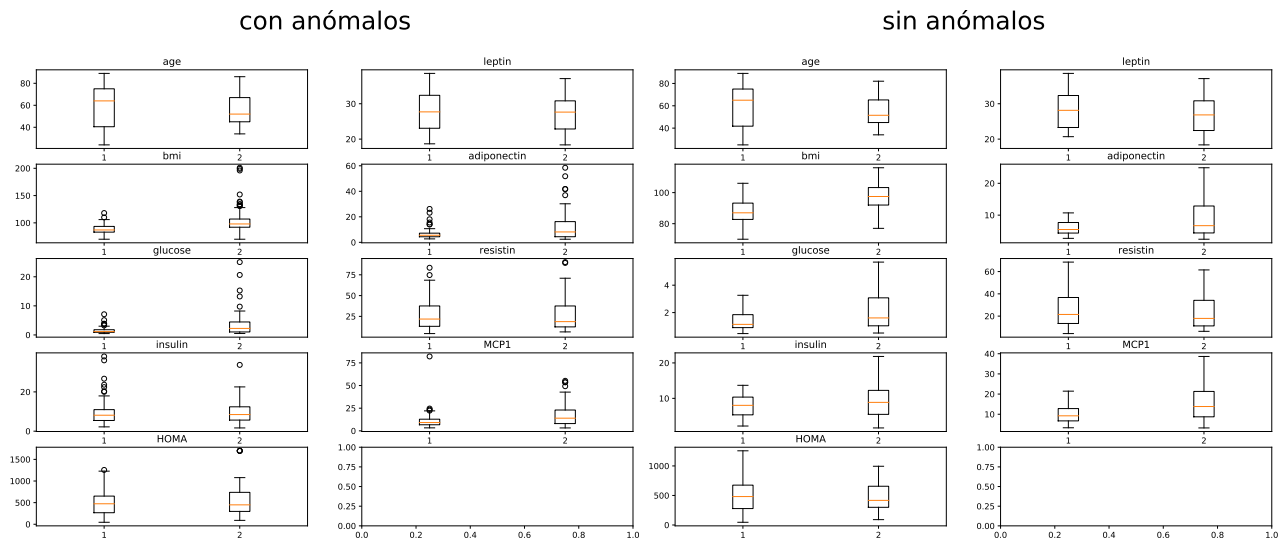


Fig. 4: Boxplots Python para datos con y sin anomalías.

```

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 5: Código Python generador de los boxplots con datos anómalos.

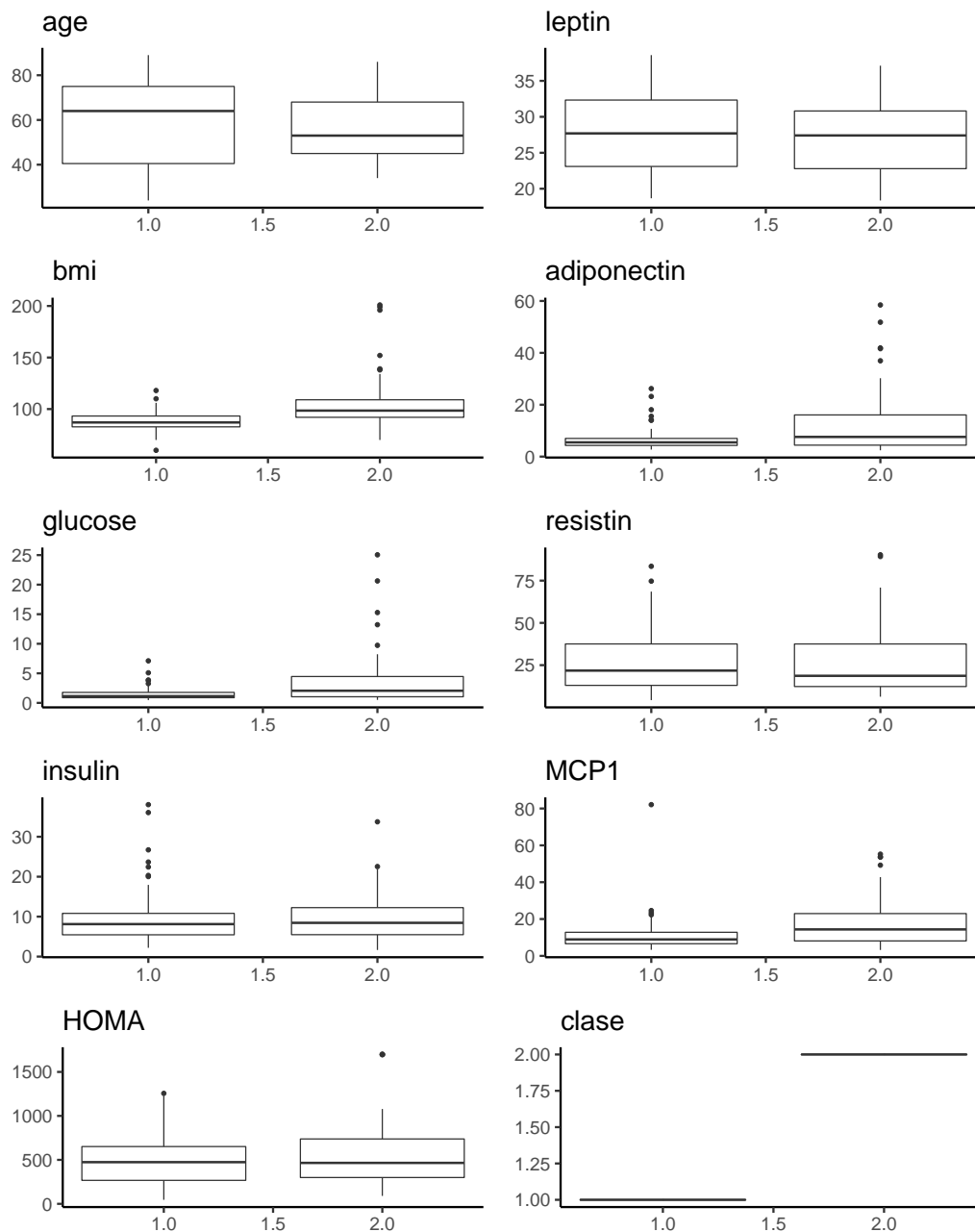


Fig. 5: Boxplots R para datos con anomalías.

```

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 6: Código R generador de los boxplots con datos anómalos.

5 QQplot

con anómalos

sin anómalos

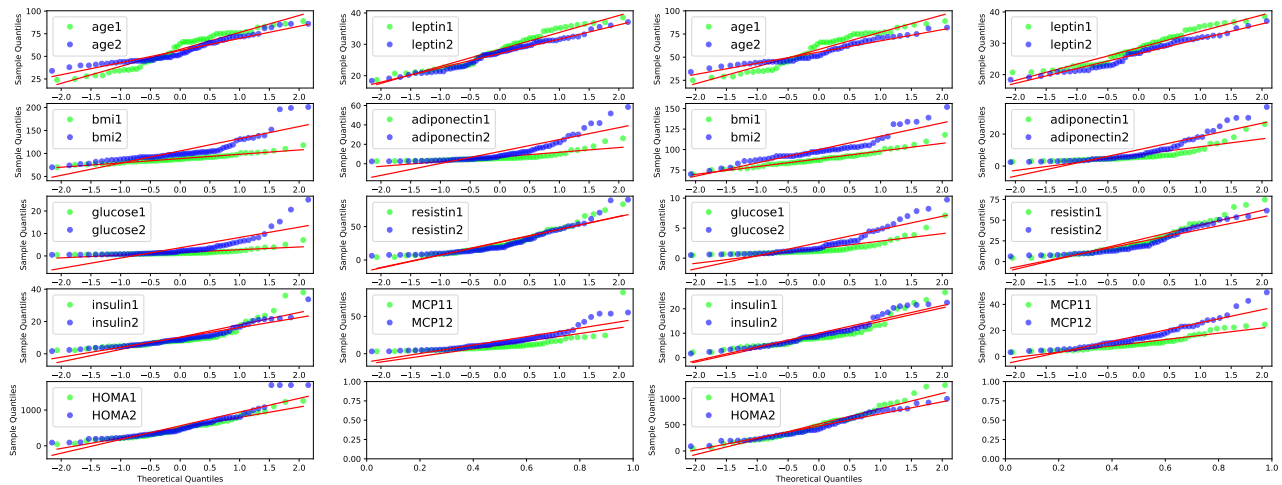


Fig. 6: QQplots para datos con y sin anomalías.

```

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 7: Código generador de los QQplots con datos anómalos.

6 Corrplot

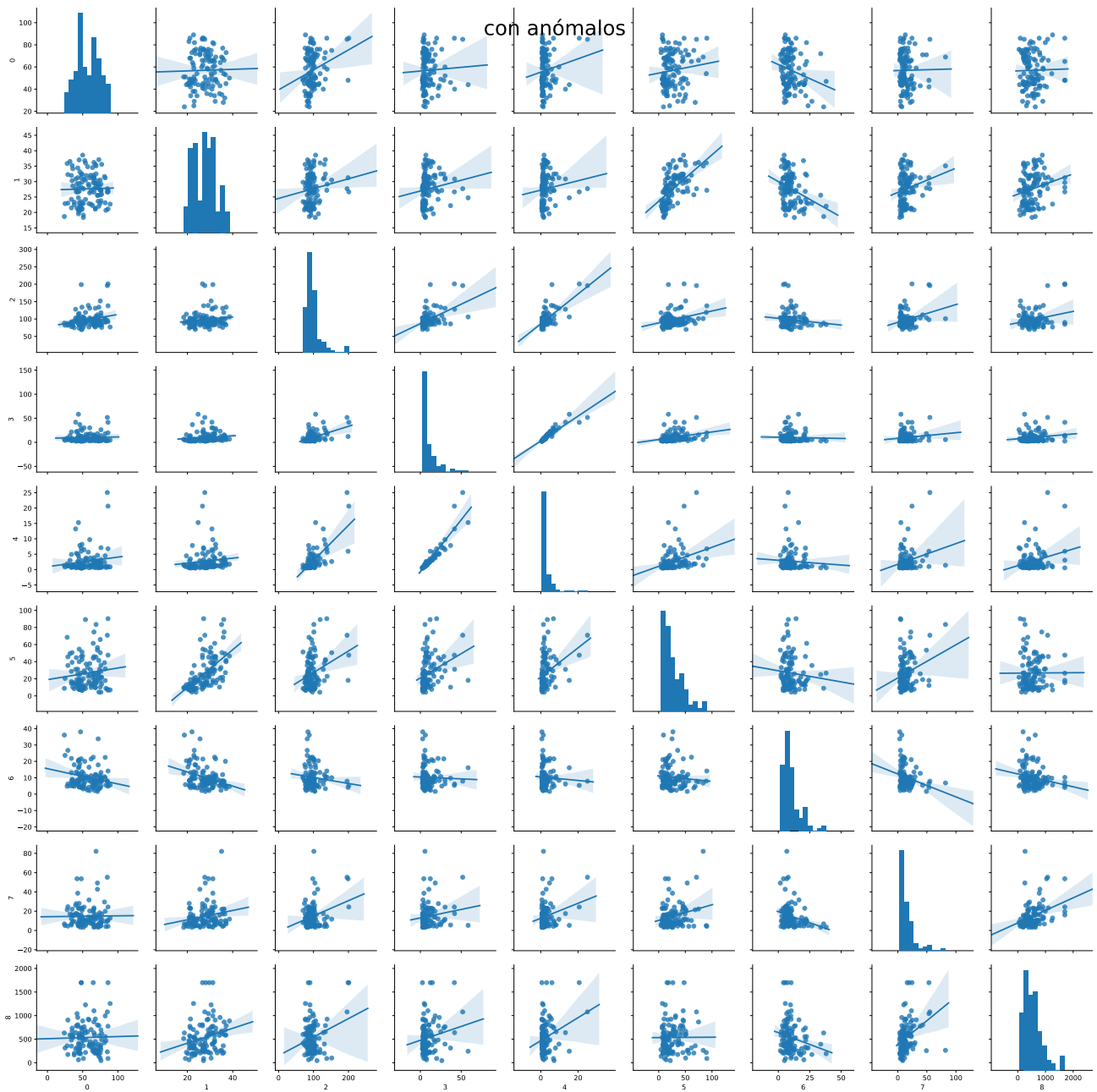


Fig. 7: Corrplot para datos con anomalías.

```
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 anómalos', fontsize = 30)
6 plt.savefig ('../images/corrp.pdf', bbox_inches = 'tight', pad_inches = 0)
```

Listing 8: Código generador de los corrplots con datos anómalos.

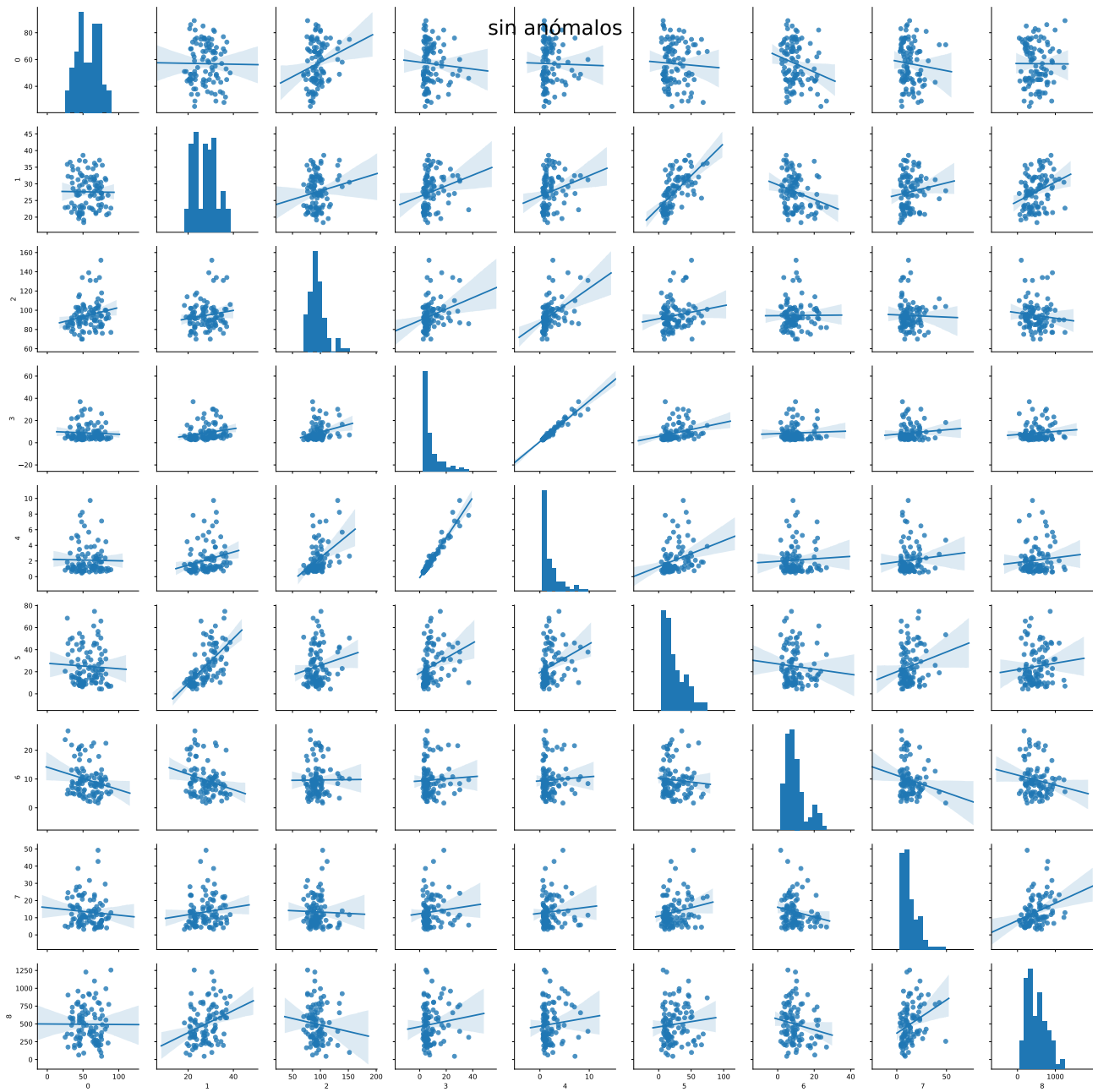


Fig. 8: Corplot para datos sin anomalías.

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 9: 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 10: Ranking de variables según los métodos filter.

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 11: 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 12: Resultados del filtrado mediante wrappers.

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 13: *Principal Component Analysis*

```
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 14: Varianza explicada por componente y suma acumulada.

9.1 Pareto

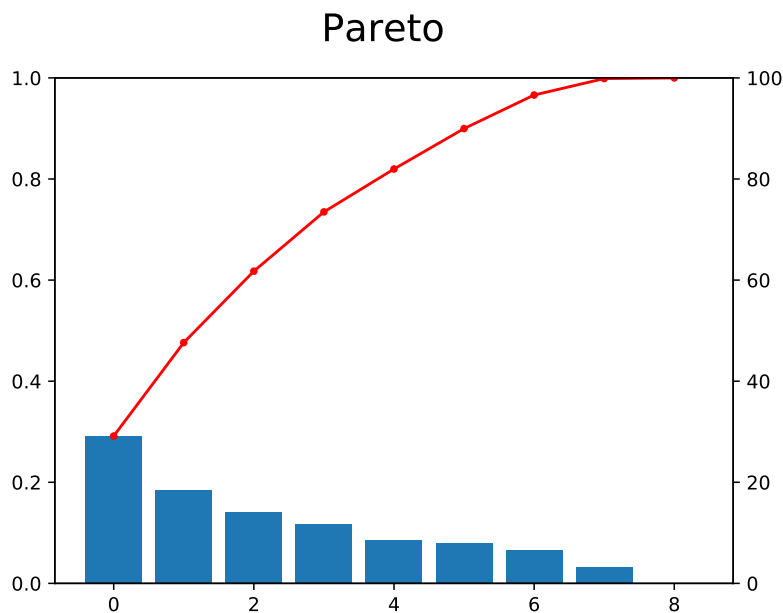


Fig. 9: Diagrama de Pareto.

```
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', fontsize = 20)
7 fig.savefig('../images/pareto.pdf', bbox_inches = 'tight', pad_inches = 0)
```

Listing 15: Código generador del diagrama de Pareto

9.2 Biplot

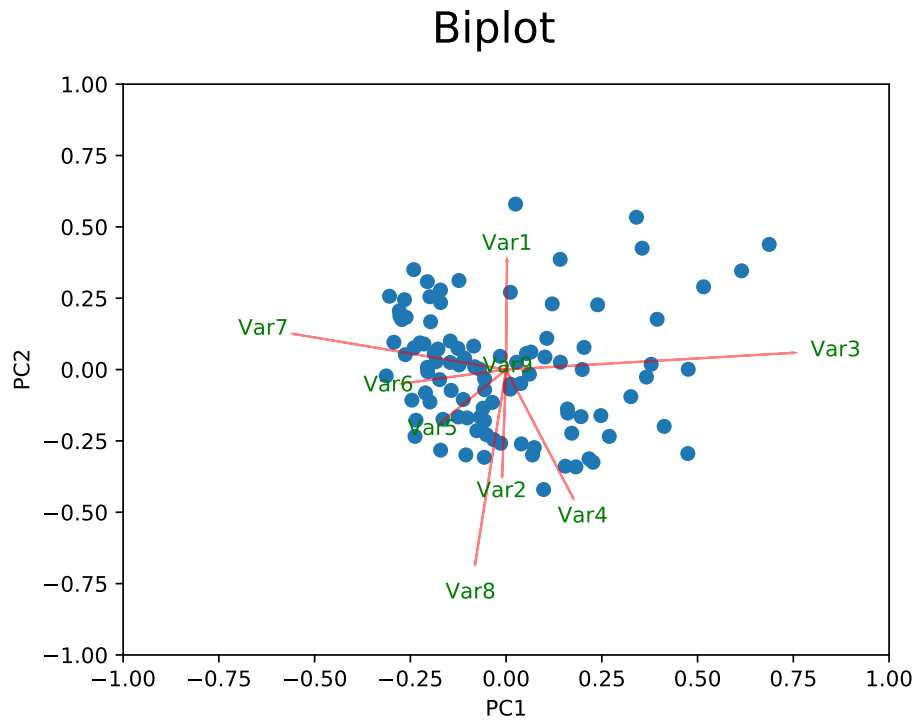


Fig. 10: Biplot.

```
1 def biplot(score,coeff,pcax,pcay,labels=None):
2     pca1=pcax-1; pca2=pcay-1
3     xs = score[:,pca1]; ys = score[:,pca2]
4     n=score.shape[1]
5     scalex = 1.0/(xs.max()- xs.min()); scaley = 1.0/(ys.max()- ys.min())
6     plt.scatter(xs*scalex,ys*scaley)
7     for i in range(n):
8         plt.arrow(0, 0, coeff[i,pca1], coeff[i,pca2],color='r',alpha=0.5)
9         if labels is None:
10             plt.text(coeff[i,pca1]* 1.15, coeff[i,pca2] * 1.15, "Var"+str(i+1), \
11                      color='g', ha='center', va='center')
12         else:
13             plt.text(coeff[i,pca1]* 1.15, coeff[i,pca2] * 1.15, labels[i], color='g', \
14                      , ha='center', va='center')
15     plt.xlim(-1,1); plt.ylim(-1,1)
16     plt.xlabel("PC{}".format(pcax)); plt.ylabel("PC{}".format(pcay))
17     return plt
18 bp = biplot (pca.fit_transform (x_no), pca.components_,1,2)
19 bp.suptitle ('Biplot', fontsize = 20)
20 bp.savefig ('../images/biplotpca.pdf', bbox_inches = 'tight', pad_inches = 0)
```

Listing 16: Código generador del Biplot.

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 17: 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 18: Validación según distintos métodos.

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 L0 media: %.2f std: %.2f'
18      %(np.mean (score), np.std (score)))
```

Listing 19: 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 L0 media: 0.73 std: 0.44
```

Listing 20: Validación según distintos métodos.

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 21: 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 22: Validación según distintos métodos.

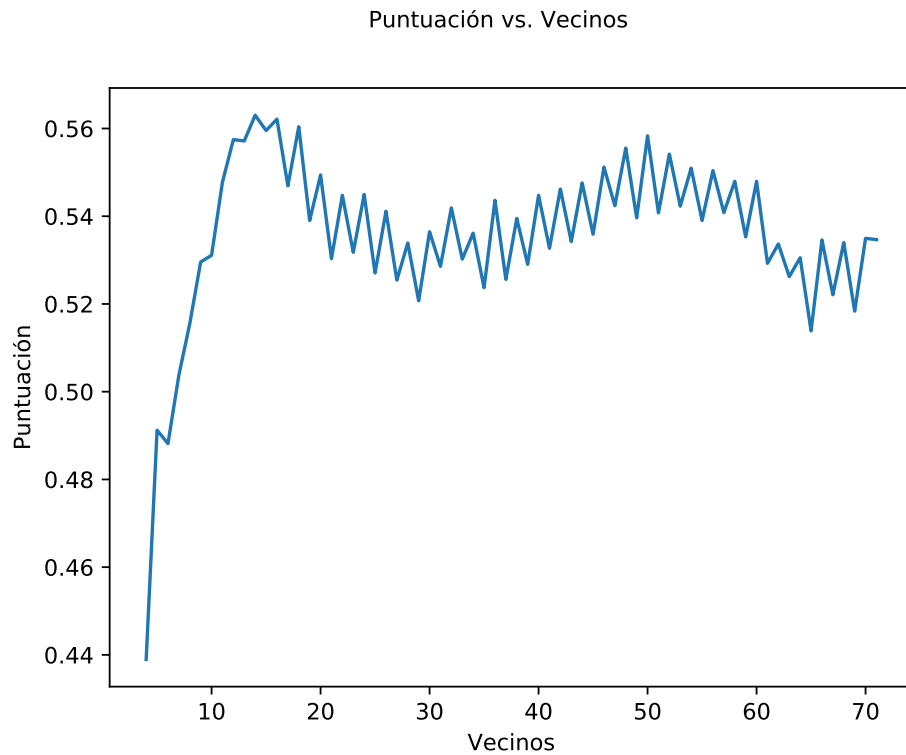


Fig. 11: Rendimiento decreciente según aumenta el número de vecinos.

```

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 23: Evolución de puntuación según número de vecinos.