Universidad Nacional Autónoma de México

Facultad de Ingeniería

PRÁCTICA 18

CLASIFICACIÓN CON SVM

Minería de Datos

Profesor:

Dr. Molero Castillo Guillermo Gilberto

Grupo 1

Alumna:

Monroy Velázquez Alejandra Sarahí

No. Cuenta: 314000417

OBJETIVO

Clasificar registros clínicos de tumores malignos y benignos de cáncer de mama a partir de imágenes digitalizadas.

DESARROLLO

El conjunto de datos con el que se trabajará corresponde a estudios clínicos a partir de imágenes digitalizadas de pacientes con cáncer de mama de Wisconsin (WDBC, Wisconsin Diagnostic Breast Cancer).

Primero comenzamos la importación de bibliotecas correspondientes que nos ayudarán para la realización del código, las cuales son *pandas* para la manipulación y análisis de datos, *numpy* para crear vectores y matrices, *matplotlib* para la generación de gráficas, así como *seaborn* para la visualización de datos. Por último, la biblioteca files para subir el archivo *csv*.

```
import pandas as pd                # Para la manipulación y análisis de datos
import numpy as np                 # Para crear vectores y matrices n dimensionales
import matplotlib.pyplot as plt    # Para la generación de gráficas a partir de los datos
import seaborn as sns              # Para la visualización de datos basado en matplotlib
%matplotlib inline
```

```
from google.colab import files
files.upload()
```

```
Elegir archivos  Ningún archivo seleccionado  Upload widget is only available when the cell has been executed in the current brow
Saving WDBCOriginal.csv to WDBCOriginal.csv
{'WDBCOriginal.csv': b'\xef\xbb\xbfIDNumber,Diagnosis,Radius,Texture,Perimeter,Area,Smoothness,Compactne
```

Una vez importadas, el *dataframe* se lee y se despliega en pantalla:

```
BCancer = pd.read_csv('WDBCOriginal.csv')
BCancer
```

	IDNumber	Diagnosis	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry	FractalDimension
0	P-842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871
1	P-842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667
2	P-84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999
3	P-84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744
4	P-84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883
...	...	...	...	...	...	...	...	...	...	...	...	...
564	P-926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623
565	P-926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533
566	P-926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648
567	P-927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016
568	P-92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884

569 rows × 12 columns

Ahora que el *dataframe* está cargado, comenzamos con los pasos vistos en clase.

Como paso extra podemos observar la columna de *Diagnosis* para ver cuántos casos fueron diagnosticados como Benignos (“B”) y cuantos como Malignos (“M”):

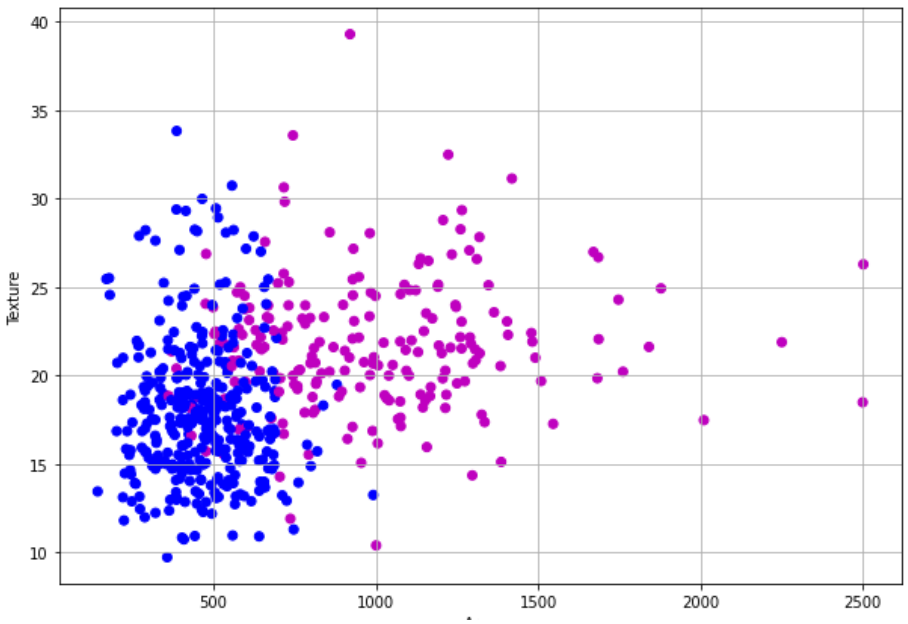
```
print(BCancer.groupby('Diagnosis').size())
```

```
Diagnosis
B    357
M    212
dtype: int64
```

A su vez, también graficamos en un plano para realizar una evaluación visual:

```
plt.figure(figsize=(10, 7))
plt.scatter(BCancer['Area'], BCancer['Texture'], c = BCancer.Diagnosis)
plt.grid()
plt.xlabel('Area')
plt.ylabel('Texture')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: MatplotlibDeprecat
```

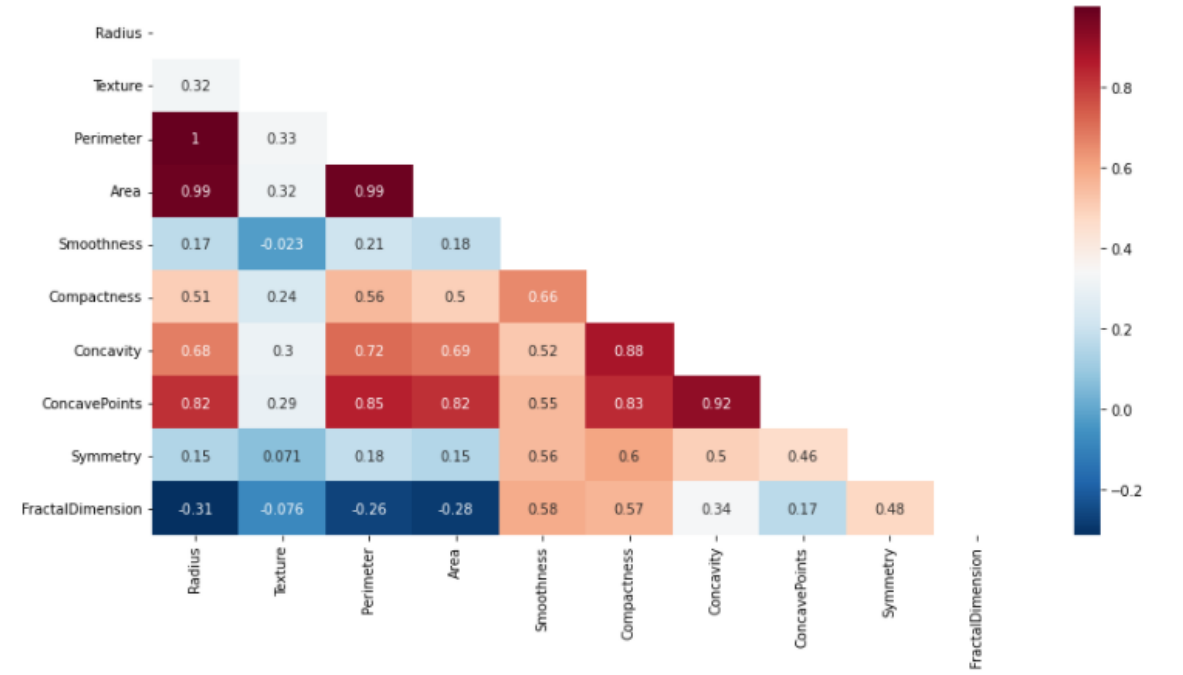


Selección de Características

Antes de pasar a la aplicación del algoritmo, se realiza una selección de caracteristicas, para descartar aquellas variables poco relevantes, o redundantes,

en este caso se hara un analisis correlacional de datos. Para ello generamos un mapa de calor mostrando solo la matriz inferior:

```
plt.figure(figsize=(14,7))
MatrizInf = np.triu(BCancer.corr())
sns.heatmap(BCancer.corr(), cmap='RdBu_r', annot=True, mask=MatrizInf)
plt.show()
```



Las variables seleccionadas son:

1) Textura [Posición 3]

2) Area [Posición 5]

3) Smoothness [Posición 6]

4) Compactness [Posición 7]

5) Symmetry [Posición 10]

6) FractalDimension [Posición 11]

Definición de variables predictoras y variable clase

Lo siguiente será seleccionar las variables predictoras (X) y la variable clase (Y), para ello primero remplazamos los términos de la columna *Diagnosis*, para que, en lugar de *B* y *M*, se les nombre como *1* y *-1* respectivamente.

```
BCancer = BCancer.replace({'M': -1, 'B': 1})
BCancer
```

	IDNumber	Diagnosis	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry	FractalDimension
0	P-842302	-1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871
1	P-842517	-1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667
2	P-84300903	-1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999
3	P-84348301	-1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744
4	P-84358402	-1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883
...	...	...	...	...	...	...	...	...	...	...	...	...
564	P-926424	-1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623
565	P-926682	-1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533
566	P-926954	-1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648
567	P-927241	-1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016
568	P-92751	1	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884

569 rows × 12 columns

```
X = np.array(BCancer[['Texture',
                      'Area',
                      'Smoothness',
                      'Compactness',
                      'Symmetry',
                      'FractalDimension']])
pd.DataFrame(X)
```

	0	1	2	3	4	5
0	10.38	1001.0	0.11840	0.27760	0.2419	0.07871
1	17.77	1326.0	0.08474	0.07864	0.1812	0.05667
2	21.25	1203.0	0.10960	0.15990	0.2069	0.05999
3	20.38	386.1	0.14250	0.28390	0.2597	0.09744
4	14.34	1297.0	0.10030	0.13280	0.1809	0.05883
...	...	...	...	...	...	...
564	22.39	1479.0	0.11100	0.11590	0.1726	0.05623
565	28.25	1261.0	0.09780	0.10340	0.1752	0.05533
566	28.08	858.1	0.08455	0.10230	0.1590	0.05648
567	29.33	1265.0	0.11780	0.27700	0.2397	0.07016
568	24.54	181.0	0.05263	0.04362	0.1587	0.05884

569 rows × 6 columns

```
Y = np.array(BCancer[['Diagnosis']])
pd.DataFrame(Y)
```

	0
0	-1
1	-1
2	-1
3	-1
4	-1
...	...
564	-1
565	-1
566	-1
567	-1
568	1

569 rows × 1 columns

Las variables predictoras serán *Texture*, *Area*, *Smoothness*, *Compactness*, *Symmetry* y *FractalDimension*, mientras que la variable clase será *Diagnosis*.

División de los datos y aplicación del algoritmo

Para comenzar a aplicar el algoritmo necesitamos importar la librería *sklearn*, ya que utilizaremos los módulos *SVC*, *classification_report*, *confusión_matrix*, *accuracy_score* y *model_selection*.

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import model_selection
```

Se hace la división de los datos donde se generará el entrenamiento y se despliega el tamaño de *X_train*, y *X_validation*:

```
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y,
                                                                                test_size = 0.2,
                                                                                random_state = 0,
                                                                                shuffle = True)
```

```
print(len(X_train))
print(len(X_validation))
```

```
455
114
```

Modelo SVM Lineal

Se declara el tipo de kernel, en este caso lineal, y se entrena el modelo:

```
ModeloSVM_1 = SVC(kernel='linear')
ModeloSVM_1.fit(X_train, Y_train)
```

Se generan las clasificaciones:

```
Clasificaciones_1 = ModeloSVM_1.predict(X_validation)
print(Clasificaciones_1)
#pd.DataFrame(Clasificaciones_1)
```

```
[ 1  1  1  1  1  1  1  1  1  1  1 -1  1 -1  1  1  1 -1 -1 -1  1 -1  1  1
 -1  1  1 -1  1 -1  1 -1  1 -1  1  1  1 -1  1 -1 -1  1 -1  1  1 -1  1  1
  1 -1 -1 -1 -1  1  1  1  1  1  1 -1  1 -1  1  1 -1  1 -1 -1 -1  1  1 -1
  1 -1 -1  1  1  1  1  1 -1 -1 -1  1 -1  1  1  1 -1 -1  1  1 -1 -1  1  1
 -1  1  1  1  1  1  1 -1 -1  1 -1  1 -1 -1  1 -1 -1  1]
```

Realizamos una comparativa entre *Y_validation* , y *Clasificaciones_1*, este último es el resultado del entrenamiento:

```
Clasificaciones = pd.DataFrame(Y_validation, Clasificaciones_1)
Clasificaciones
```

	0
1	-1
1	1
1	1
1	1
1	1
...	...
-1	-1
1	1
-1	-1
-1	-1
1	1

114 rows × 1 columns

Se calcula la exactitud promedio de la validación:

```
ModeloSVM_1.score(X_validation, Y_validation)
```

```
0.9122807017543859
```

Validación del modelo

Validamos el modelo a través de una matriz de clasificación:

```
Clasificaciones_1 = ModeloSVM_1.predict(X_validation)
Matriz_Clasificacion = pd.crosstab(Y_validation.ravel(),
                                   Clasificaciones_1,
                                   rownames=['Real'],
                                   colnames=['Clasificación'])
Matriz_Clasificacion
```

Clasificación	-1	1
Real		
-1	41	6
1	4	63

Observamos que, de 47 casos malignos, el modelo clasifico erróneamente 6 de ellos, y de 67 casos benignos, el modelo clasifico erróneamente solo 4.

Generamos un reporte de clasificación:

```
print("Exactitud", ModeloSVM_1.score(X_validation, Y_validation))
print(classification_report(Y_validation, Clasificaciones_1))
```

```
Exactitud 0.9122807017543859
              precision    recall  f1-score   support

          -1       0.91      0.87      0.89        47
           1       0.91      0.94      0.93        67

    accuracy                           0.91       114
   macro avg       0.91      0.91      0.91       114
weighted avg       0.91      0.91      0.91       114
```

Generamos e imprimimos los vectores de soporte:

```
VectoresSoporte_1 = ModeloSVM_1.support_vectors_
pd.DataFrame(VectoresSoporte_1)
```

	0	1	2	3	4	5
0	23.12	609.9	0.10750	0.24130	0.2384	0.07542
1	19.07	701.9	0.09215	0.08597	0.1561	0.05915
2	23.94	668.3	0.11720	0.14790	0.1953	0.06654
3	24.69	572.6	0.09258	0.07862	0.1761	0.06130
4	19.97	744.7	0.11600	0.15620	0.1929	0.06744
...	...	...	...	...	...	...
116	19.34	659.7	0.08388	0.07800	0.1473	0.05746
117	24.44	406.4	0.12360	0.15520	0.2131	0.07405
118	16.85	666.0	0.08641	0.06698	0.1409	0.05355
119	18.89	558.1	0.10590	0.11470	0.1806	0.06079
120	14.93	686.9	0.08098	0.08549	0.1687	0.05669

121 rows × 6 columns

```
print('Número de vectores de soporte: \n', ModeloSVM_1.n_support_)
print('Vectores de soporte: \n', ModeloSVM_1.support_)
```

```
Número de vectores de soporte:
 [60 61]
Vectores de soporte:
 [  8  23  41  43  45  52  65  69  78  85 106 108 113 123 131 134 135 138
 142 165 177 179 182 186 208 214 216 223 224 228 241 247 250 259 266 282
 289 290 315 321 328 333 336 338 339 341 354 362 381 400 401 403 406 408
 411 413 421 435 436 451   2  19  22  37  39  54  55  68  72 103 110 116
 141 149 153 157 168 171 176 192 195 207 209 210 213 217 218 230 231 233
 236 239 244 264 268 278 284 287 299 312 314 322 335 345 346 356 384 386
 396 397 410 422 429 430 433 434 437 441 445 447 448]
```

Modelo SVM Polinomial

Se declara el tipo de kernel, en este caso polinomial, y se entrena el modelo:

```
ModeloSVM_2 = SVC(kernel='poly', degree=3)
ModeloSVM_2.fit(X_train, Y_train)
```

Se generan las clasificaciones:

```
Clasificaciones_2 = ModeloSVM_2.predict(X_validation)
print(Clasificaciones_2)
#pd.DataFrame(Clasificaciones_2)
```

```
[ 1  1  1  1  1  1  1  1  1  1  1  1  1 -1  1 -1  1 -1 -1 -1  1 -1  1  1
 -1  1  1  1  1 -1  1 -1  1 -1  1  1  1 -1  1  1  1  1 -1  1  1 -1  1  1
  1  1 -1  1 -1  1  1  1  1  1  1 -1  1 -1  1  1 -1  1 -1 -1 -1  1  1 -1
  1  1 -1  1  1  1  1  1 -1 -1 -1  1 -1  1  1  1 -1 -1  1  1  1  1  1  1
 -1  1  1  1  1  1  1  1 -1  1 -1  1 -1 -1  1 -1 -1  1]
```

Se calcula la exactitud promedio de la validación:

```
ModeloSVM_2.score(X_validation, Y_validation)
```

```
0.8859649122807017
```

Se probó con otros grados de polinomio, pero el que mejor resultado dio fue el grado tres, el que es por defecto, por lo cual mantenemos este valor.

Validación del modelo

Validamos el modelo a través de una matriz de clasificación:

```
Clasificaciones_2 = ModeloSVM_2.predict(X_validation)
Matriz_Clasificacion = pd.crosstab(Y_validation.ravel(),
                                   Clasificaciones_2,
                                   rownames=['Real'],
                                   colnames=['Clasificación'])
Matriz_Clasificacion
```

Clasificación	-1	1
Real		
-1	35	12
1	1	66

Observamos que, de 47 casos malignos, el modelo clasifico erróneamente 12 de ellos, y de 67 casos benignos, el modelo clasifico erróneamente solo 1.

Generamos un reporte de clasificación:

```
print("Exactitud", ModeloSVM_2.score(X_validation, Y_validation))
print(classification_report(Y_validation, Clasificaciones_2))
```

```
Exactitud 0.8859649122807017
              precision    recall  f1-score   support

          -1       0.97      0.74      0.84        47
           1       0.85      0.99      0.91        67

    accuracy                           0.89       114
   macro avg       0.91      0.86      0.88       114
weighted avg       0.90      0.89      0.88       114
```

Generamos e imprimimos los vectores de soporte:

```
VectoresSoporte_2 = ModeloSVM_2.support_vectors_
pd.DataFrame(VectoresSoporte_2)
```

	0	1	2	3	4	5
0	23.12	609.9	0.10750	0.24130	0.2384	0.07542
1	19.07	701.9	0.09215	0.08597	0.1561	0.05915
2	23.94	668.3	0.11720	0.14790	0.1953	0.06654
3	24.69	572.6	0.09258	0.07862	0.1761	0.06130
4	19.97	744.7	0.11600	0.15620	0.1929	0.06744
...	...	...	...	...	...	...
131	19.13	575.3	0.09057	0.11470	0.1848	0.06181
132	18.77	689.5	0.08138	0.11670	0.1744	0.06493
133	19.34	659.7	0.08388	0.07800	0.1473	0.05746
134	16.85	666.0	0.08641	0.06698	0.1409	0.05355
135	14.93	686.9	0.08098	0.08549	0.1687	0.05669

136 rows × 6 columns

```
print('Número de vectores de soporte: \n', ModeloSVM_2.n_support_)
print('Vectores de soporte: \n', ModeloSVM_2.support_)
```

```
Número de vectores de soporte:
 [68 68]
Vectores de soporte:
 [  8  23  41  43  45  52  65  69  78  85 106 108 113 123 131 133 134 135
 136 138 142 165 177 179 182 186 208 214 216 223 224 241 247 250 254 259
 266 273 282 289 290 308 315 321 328 332 333 336 338 339 341 354 361 362
 363 381 400 401 403 406 408 411 413 421 423 435 436 451   2  12  22  32
  35  37  42  54  68  72  84 103 105 110 116 118 141 143 153 159 168 171
 184 185 192 195 210 213 217 218 225 230 233 236 244 264 275 283 287 302
 307 310 312 314 317 322 335 342 345 346 351 356 373 384 386 394 395 396
 397 410 414 422 430 433 434 437 445 448]
```

Modelo SVM RBF (Función de Base Radial)

Se declara el tipo de kernel, en este caso RBF, y se entrena el modelo:

```
ModeloSVM_3 = SVC(kernel='rbf')
ModeloSVM_3.fit(X_train, Y_train)
```

Se generan las clasificaciones:

```
Clasificaciones_3 = ModeloSVM_3.predict(X_validation)
print(Clasificaciones_3)
```

```
[ 1  1  1  1  1  1  1  1  1  1  1  1 -1  1  1  1 -1 -1 -1  1 -1  1  1
 -1  1  1  1  1 -1  1 -1  1 -1  1  1  1 -1  1  1  1  1 -1  1  1 -1  1  1
  1  1 -1  1 -1  1  1  1  1  1  1 -1  1 -1  1  1 -1  1 -1 -1 -1  1  1 -1
  1  1 -1  1  1  1  1  1 -1 -1 -1  1 -1  1  1  1 -1 -1  1  1  1  1  1  1
 -1  1  1  1  1  1  1  1 -1  1 -1  1 -1 -1  1 -1 -1  1]
```

Se calcula la exactitud promedio de la validación:

```
ModeloSVM_3.score(X_validation, Y_validation)
```

```
0.8771929824561403
```

Validación del modelo

Validamos el modelo a través de una matriz de clasificación:

```
Clasificaciones_3 = ModeloSVM_3.predict(X_validation)
Matriz_Clasificacion = pd.crosstab(Y_validation.ravel(),
                                   Clasificaciones_3,
                                   rownames=['Real'],
                                   colnames=['Clasificación'])
Matriz_Clasificacion
```

Clasificación	-1	1
Real		
-1	34	13
1	1	66

Observamos que, de 47 casos malignos, el modelo clasifico erróneamente 13 de ellos, y de 67 casos benignos, el modelo clasifico erróneamente solo 1.

Generamos un reporte de clasificación:

```
print("Exactitud", ModeloSVM_3.score(X_validation, Y_validation))
print(classification_report(Y_validation, Clasificaciones_3))
```

```
Exactitud 0.8771929824561403
              precision    recall  f1-score   support

          -1       0.97      0.72      0.83        47
           1       0.84      0.99      0.90        67

    accuracy                           0.88       114
   macro avg       0.90      0.85      0.87       114
weighted avg       0.89      0.88      0.87       114
```

Generamos e imprimimos los vectores de soporte:

```
VectoresSoporte_3 = ModeloSVM_3.support_vectors_
pd.DataFrame(VectoresSoporte_3)
```

	0	1	2	3	4	5
0	23.12	609.9	0.10750	0.24130	0.2384	0.07542
1	19.07	701.9	0.09215	0.08597	0.1561	0.05915
2	20.20	857.6	0.07497	0.07112	0.1846	0.05325
3	23.94	668.3	0.11720	0.14790	0.1953	0.06654
4	24.69	572.6	0.09258	0.07862	0.1761	0.06130
...	...	...	...	...	...	...
140	18.77	689.5	0.08138	0.11670	0.1744	0.06493
141	19.34	659.7	0.08388	0.07800	0.1473	0.05746
142	16.85	666.0	0.08641	0.06698	0.1409	0.05355
143	18.89	558.1	0.10590	0.11470	0.1806	0.06079
144	14.93	686.9	0.08098	0.08549	0.1687	0.05669

145 rows × 6 columns

```
print('Número de vectores de soporte: \n', ModeloSVM_3.n_support_)
print('Vectores de soporte: \n', ModeloSVM_3.support_)
```

```
Número de vectores de soporte:
 [73 72]
Vectores de soporte:
 [  8  23  33  41  43  45  51  52  65  69  78  85 106 108 113 123 126 131
 133 134 135 136 138 142 165 177 179 182 186 208 214 216 223 224 226 241
 247 250 254 259 266 273 282 289 290 308 315 321 328 332 333 336 338 339
 341 354 361 362 363 370 381 400 401 403 406 408 411 413 421 423 435 436
 451   2  12  22  27  32  35  37  42  54  61  68  72  84 103 105 110 116
 118 141 143 153 159 168 171 184 185 192 195 210 213 217 218 225 230 233
 236 244 264 268 275 283 287 302 307 310 312 314 317 322 335 342 345 346
 351 356 373 384 386 394 395 396 397 410 414 422 430 433 434 437 445 447
 448]
```

Modelo SVM Sigmoide

Se declara el tipo de kernel, en este caso Sigmoide, y se entrena el modelo:

```
ModeloSVM_4 = SVC(kernel='sigmoid')
ModeloSVM_4.fit(X_train, Y_train)
```

Se generan las clasificaciones:

```
Clasificaciones_4 = ModeloSVM_4.predict(X_validation)
print(Clasificaciones_4)
```

```
[ 1  1  1  1 -1  1 -1  1  1  1  1  1  1  1  1  1  1  1  1 -1  1  1  1
  1 -1  1  1  1  1 -1  1  1  1  1 -1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1 -1  1  1 -1 -1  1  1  1  1 -1  1 -1  1  1  1  1  1  1
  1  1  1 -1  1 -1  1  1  1  1  1 -1  1 -1  1  1  1  1  1  1  1  1  1  1
  1 -1  1  1  1 -1  1  1  1  1  1 -1  1  1 -1  1  1  1]
```

Se calcula la exactitud promedio de la validación:

```
ModeloSVM_4.score(X_validation, Y_validation)
```

```
0.45614035087719296
```

En este caso ya no procedemos a la validación del modelo, ya que la exactitud es bastante mala, menos del 50%, por lo que desde este momento descartamos al modelo.

Nuevas clasificaciones

Para nuevas clasificaciones implementamos el siguiente bloque de código, donde cada variable predictora tiene un valor; esto servirá para clasificar el valor de la variable clase, es decir del *diagnóstico:*

```
PacienteID111 = pd.DataFrame({'Texture': [10.38],
                              'Area': [1001.0],
                              'Smoothness': [0.11840],
                              'Compactness': [0.27760],
                              'Symmetry': [0.2419],
                              'FractalDimension': [0.07871]})
ModeloSVM_1.predict(PacienteID111)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:439: Us
  f"X has feature names, but {self.__class__.__name__} was fit
array([-1])
```

Cabe recalcar que para realizar esta clasificación tomamos el modelo que mejor resultado nos dio, el cual fue el lineal, ya que obtuvo 91% de exactitud, la más alta con respecto a los demás modelos.

CONCLUSIÓN

En esta practica utilizamos máquina de soporte vectorial para entrenar diversos modelos, probando el modelo lineal, polinomial, rbf y sigmoide, siendo el primero de estos el que mejor exactitud nos produjo, por lo que ese mismo se utilizo para realizar nuevas clasificaciones.