Universidad Nacional Autónoma de México

Facultad de Ingeniería

PRÁCTICA 17

CLASIFICACIÓN (BOSQUES ALEATORIOS)

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”):

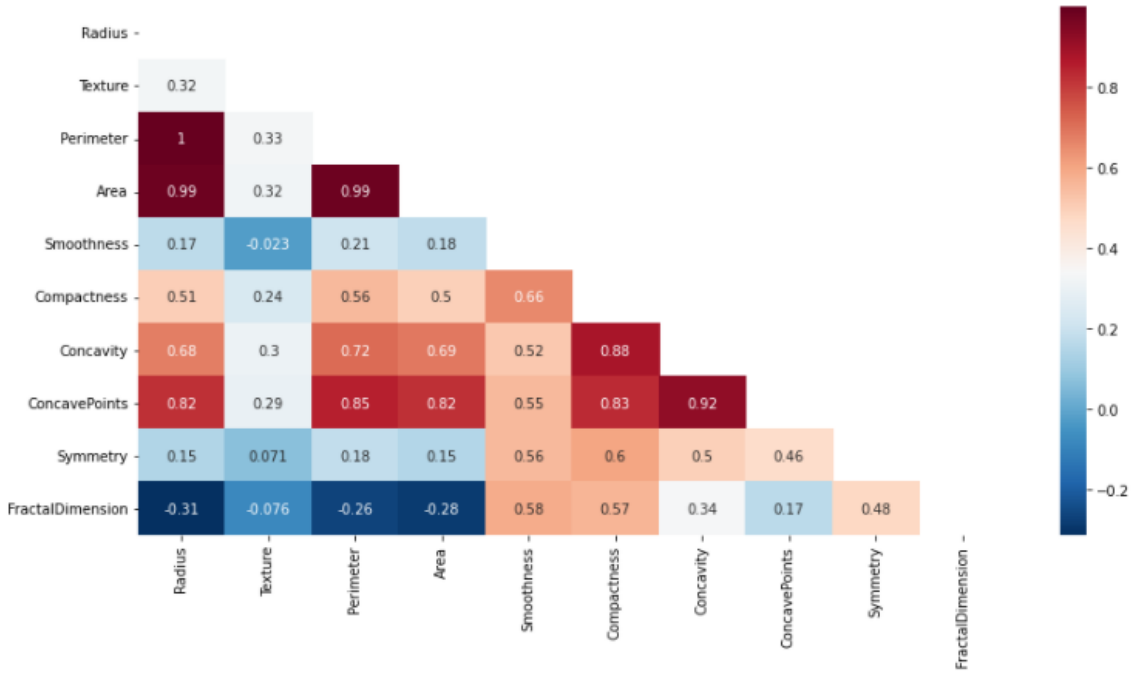
```
print(BCancer.groupby('Diagnosis').size())
```

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

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 *Benign* y *Malignant* respectivamente.

```
BCancer = BCancer.replace({'M': 'Malignant', 'B': 'Benign'})
BCancer
```

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

#X = np.array(BCancer[['Radius', 'Texture', 'Perimet
#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	Malignant
1	Malignant
2	Malignant
3	Malignant
4	Malignant
...	...
564	Malignant
565	Malignant
566	Malignant
567	Malignant
568	Benign

569 rows × 1 columns

División de los datos y aplicación del algoritmo

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

```
from sklearn.ensemble import RandomForestClassifier
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 tanto la variable *X_train*, como *Y_train:*

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

```
pd.DataFrame(X_train)
```

	0	1	2	3	4	5
0	17.53	310.8	0.10070	0.07326	0.1890	0.06331
1	21.98	359.9	0.08801	0.05743	0.2016	0.05977
2	14.86	800.0	0.09495	0.08501	0.1735	0.05875
3	17.84	451.1	0.10450	0.07057	0.1900	0.06635
4	22.44	466.5	0.08192	0.05200	0.1544	0.05976
...	...	...	...	...	...	...
450	19.98	1102.0	0.08923	0.05884	0.1550	0.04996
451	24.04	475.9	0.11860	0.23960	0.2030	0.08243
452	18.32	278.6	0.10090	0.05956	0.1506	0.06959
453	18.22	288.1	0.06950	0.02344	0.1653	0.06447
454	23.93	403.5	0.09261	0.10210	0.1388	0.06570

455 rows × 6 columns

```
pd.DataFrame(Y_train)
```

	0
0	Benign
1	Benign
2	Benign
3	Benign
4	Benign
...	...
450	Malignant
451	Malignant
452	Benign
453	Benign
454	Benign

455 rows × 1 columns

Luego, se entrena el modelo a través de un bosque aleatorio:

```
ClasificacionBA = RandomForestClassifier(random_state=0)
ClasificacionBA.fit(X_train, Y_train)
```

En este paso también podemos definir los parámetros *max_depth, min_samples_split, min_samples_leaf* y *random_state* para evitar un sobreajuste.

Y se generan las clasificaciones :

```
Y_Clasificacion = ClasificacionBA.predict(X_validation)
pd.DataFrame(Y_Clasificacion)
```

	0
0	Malignant
1	Benign
2	Benign
3	Benign
4	Benign
...	...
109	Malignant
110	Benign
111	Malignant
112	Malignant
113	Benign

114 rows × 1 columns

Podemos visualizar una comparativa entre el valor del diagnóstico que agrupamos en *Y_validation* contra el valor de *Y_Clasificacion*, el cual resulta del entrenamiento a través del bosque aleatorio. Si observamos cada registro, nos damos cuenta de que los valores de la clasificación se acercan a los reales, salvo algunas excepciones:

```
Valores = pd.DataFrame(Y_validation, Y_Clasificacion)
Valores
```

	0
Malignant	Malignant
Benign	Benign
Benign	Benign
Benign	Benign
Benign	Benign
...	...
Malignant	Malignant
Benign	Benign
Malignant	Malignant
Malignant	Malignant
Benign	Benign

114 rows × 1 columns

Si imprimimos nuestro *score* nos daremos cuenta de que nuestro modelo ha sido entrenado con 93% de exactitud:

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

```
0.9385964912280702
```

Validación del modelo

Se realiza una validación a través de una matriz de clasificación:

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

Clasificación	Benign	Malignant
Real		
Benign	64	3
Malignant	4	43

Luego, realizamos un reporte de clasificación, donde incluimos el criterio, la importancia de las variables y la exactitud del modelo. También imprimimos las variables y las ordenamos de acuerdo con su importancia:

```
print('Criterio: \n', ClasificacionBA.criterion)
print('Importancia variables: \n', ClasificacionBA.feature_importances_)
print("Exactitud", ClasificacionBA.score(X_validation, Y_validation))
print(classification_report(Y_validation, Y_Clasificacion))
```

```
Criterio:
 gini
Importancia variables:
 [0.12681465 0.45987493 0.07943492 0.21421778 0.05473754 0.06492018]
Exactitud 0.9385964912280702
              precision    recall  f1-score   support

      Benign       0.94      0.96      0.95        67
   Malignant       0.93      0.91      0.92        47

    accuracy                           0.94       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.94      0.94      0.94       114
```

```
Importancia = pd.DataFrame({'Variable': list(BCancer[['Texture', 'Area', 'Smoothness',
                                                      'Compactness', 'Symmetry', 'FractalDimension']]),
                            'Importancia': ClasificacionBA.feature_importances_}).sort_values('Importancia', ascending=False)
Importancia
```

	Variable	Importancia
1	Area	0.459875
3	Compactness	0.214218
0	Texture	0.126815
2	Smoothness	0.079435
5	FractalDimension	0.064920
4	Symmetry	0.054738

Eficiencia y conformación del modelo de clasificación

En la matriz de confusión se utilizó 114 instancias de prueba, clasificándose de manera errónea 7 casos. Esto hace que el modelo tenga un 93.85% de exactitud y un 94% de precisión para los casos Benignos y 93% para los casos malignos.

Por otro lado, el error promedio es de 6.15%.

Para graficar el árbol, se puede utilizar alguno de los estimadores

En este ejemplo utilizaremos el árbol número 10, pero se podría ver cualquier otro árbol con solo cambiar el número:

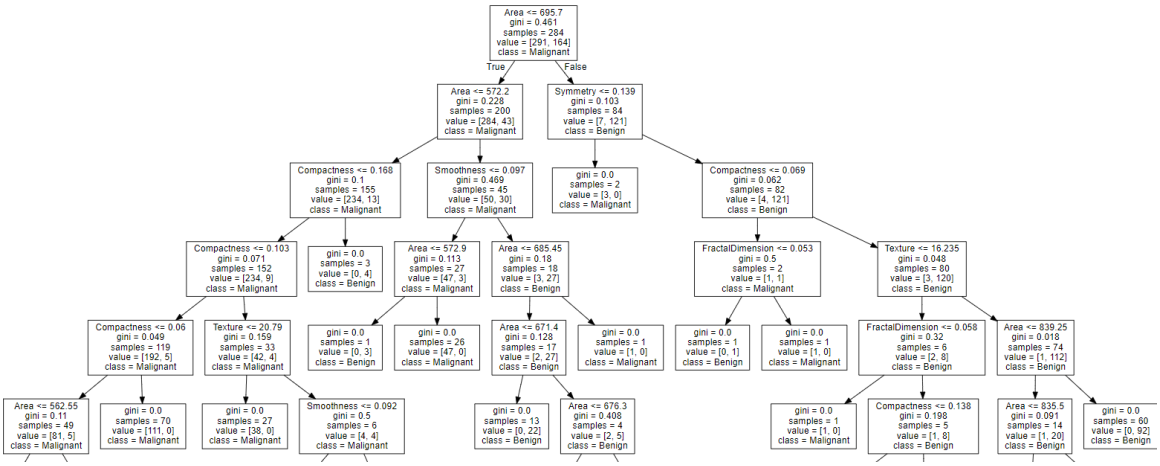
```
Estimador = ClasificacionBA.estimators_[10]
Estimador
```

```
DecisionTreeClassifier(max_features='auto', random_state=626610453)
```

Lo siguiente será instalar el paquete *graphviz* importar la biblioteca y el módulo *export_graphviz* de *sklearn* ya que imprimiremos la forma del árbol.

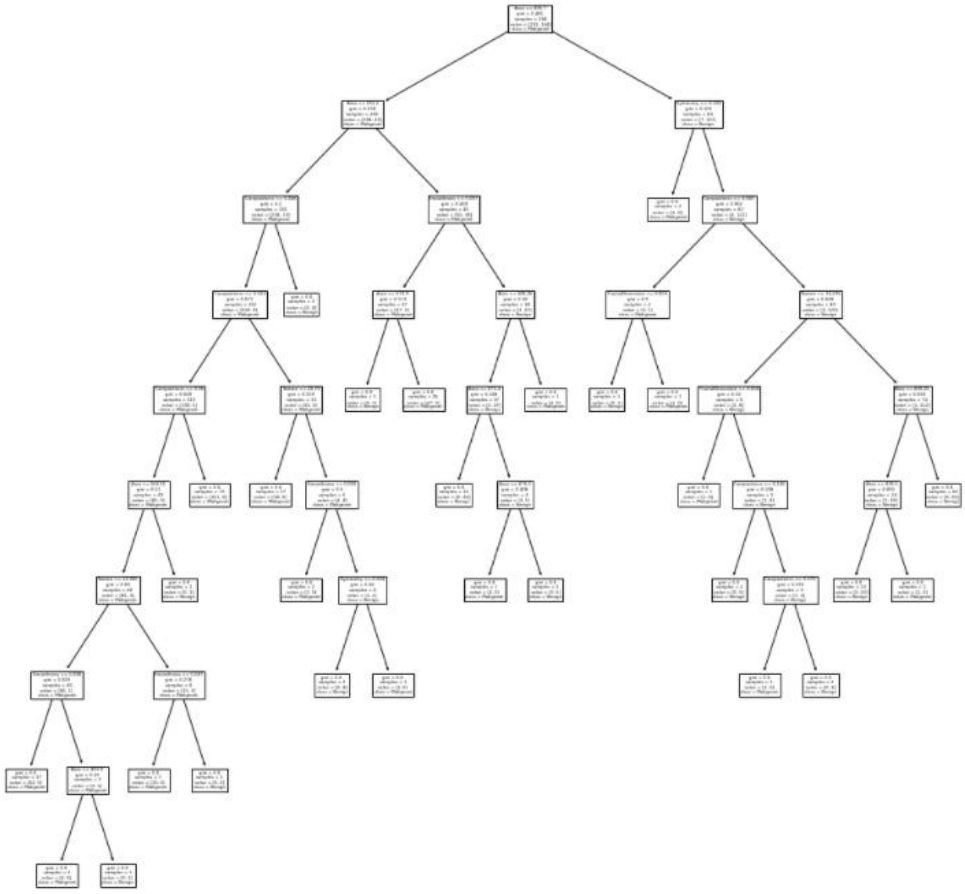
```
import graphviz
from sklearn.tree import export_graphviz
```

```
# Se crea un objeto para visualizar el árbol
Elementos = export_graphviz(Estimador,
                            feature_names = ['Texture', 'Area', 'Smoothness', 'Compactness',
                                             'Symmetry', 'FractalDimension'],
                            class_names = Y_Clasificacion)
Arbol = graphviz.Source(Elementos)
Arbol
```



Como la imagen del árbol es muy grande para visualizarla de forma general, imprimimos de forma distinta:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(16,16))
plot_tree(Estimador,
          feature_names = ['Texture', 'Area', 'Smoothness', 'Compactness',
                           'Symmetry', 'FractalDimension'],
          class_names = Y_Clasificacion)
plt.show()
```



Para una mejor legibilidad imprimimos de esta manera:

```
from sklearn.tree import export_text
Reporte = export_text(Estimador,
                      feature_names = ['Texture', 'Area', 'Smoothness', 'Compactness',
                                       'Symmetry', 'FractalDimension'])
print(Reporte)
```

```
|--- Area <= 695.70
|   |--- Area <= 572.20
|   |   |--- Compactness <= 0.17
|   |   |   |--- Compactness <= 0.10
|   |   |   |   |--- Compactness <= 0.06
|   |   |   |   |   |--- Area <= 562.55
|   |   |   |   |   |   |--- Texture <= 22.45
|   |   |   |   |   |   |   |--- Smoothness <= 0.10
|   |   |   |   |   |   |   |   |--- class: 0.0
|   |   |   |   |   |   |   |--- Smoothness >  0.10
|   |   |   |   |   |   |   |   |--- Area <= 401.90
|   |   |   |   |   |   |   |   |   |--- class: 0.0
|   |   |   |   |   |   |   |   |--- Area >  401.90
|   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- Texture >  22.45
|   |   |   |   |   |   |   |--- Smoothness <= 0.09
|   |   |   |   |   |   |   |   |--- class: 0.0
|   |   |   |   |   |   |   |--- Smoothness >  0.09
|   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |--- Area >  562.55
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- Compactness >  0.06
|   |   |   |   |   |--- class: 0.0
|   |   |   |--- Compactness >  0.10
|   |   |   |   |--- Texture <= 20.79
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- Texture >  20.79
|   |   |   |   |   |--- Smoothness <= 0.09
|   |   |   |   |   |   |--- class: 0.0
|   |   |   |   |   |--- Smoothness >  0.09
|   |   |   |   |   |   |--- Symmetry <= 0.20
|   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |--- Symmetry >  0.20
|   |   |   |   |   |   |   |--- class: 0.0
|   |   |--- Compactness >  0.17
|   |   |   |--- class: 1.0
|   |--- Area >  572.20
```

La imagen del árbol es grande, pero se puede leer en el siguiente orden:

1. La decisión que se toma para dividir el nodo.
2. El tipo de criterio que se usó para dividir cada nodo.
3. Cuantos valores tiene ese nodo.
4. Valores promedio.
5. Por último, el valor pronosticado en ese nodo.

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:*

```
#Paciente P-842302 (1) -Tumor Maligno-
PacienteID1 = pd.DataFrame({'Texture': [10.38],
                            'Area': [1001.0],
                            'Smoothness': [0.11840],
                            'Compactness': [0.27760],
                            'Symmetry': [0.2419],
                            'FractalDimension': [0.07871]})
ClasificacionBA.predict(PacienteID1)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:439: User
  f"X has feature names, but {self.__class__.__name__} was fitte
array(['Malignant'], dtype=object)
```

```
#Paciente P-92751 (569) -Tumor Benigno-
PacienteID2 = pd.DataFrame({'Texture': [24.54],
                            'Area': [181.0],
                            'Smoothness': [0.05263],
                            'Compactness': [0.04362],
                            'Symmetry': [0.1587],
                            'FractalDimension': [0.05884]})
ClasificacionBA.predict(PacienteID2)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:439: User
  f"X has feature names, but {self.__class__.__name__} was fitte
array(['Benign'], dtype=object)
```

CONCLUSIÓN

En esta práctica de nuevo aplicamos el algoritmo de bosques aleatorios, ahora haciendo una clasificación, aunque si hacemos una comparación con la practica quince, no hay mucha diferencia en cuanto al resultado de los nuevos pronósticos, pero en cuanto al entrenamiento del modelo sí que lo hay, ya que al implementar un bosque en lugar de un solo árbol se obtienen mejores resultados.