# Universidad Nacional Autónoma de México

## Inteligencia Artificial

# Bosques Aleatorios (Clasificación)

**Grupo 3**


*Nombre:*

Barreiro Valdez Alejandro


## Práctica 14

**Profesor: Dr. Guillermo Gilberto Molero Castillo**

12 de mayo de 2022

## Introducción

En esta práctica se utilizará clasificación de bosques aleatorios para predecir el área del tumor de pacientes con indicios de cáncer de mama. Se compararán los resultados con los obtenidos en prácticas pasadas utilizando pronósticos de un árbol. Además, se obtendrá la estructura de un árbol que conforma parte del bosque para saber cómo se conforma. Se generarán diferentes modelos cambiando el número de estimadores y otros parámetros para comparar la eficiencia de cada modelo.

## Objetivos

Pronosticar el área del tumor de pacientes con indicios de casos de cáncer de mama a través de bosques aleatorios utilizando estudios clínicos a partir de imágenes digitalizadas de pacientes con cáncer de mama de Wisconsin (WDBC, Wisconsin Diagnostic Breast Cancer).

## Desarrollo

Se importan las bibliotecas necesarias para la manipulación de datos.

```
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
```

Se importan los datos del cáncer de mama para utilizarlos.

```
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 |

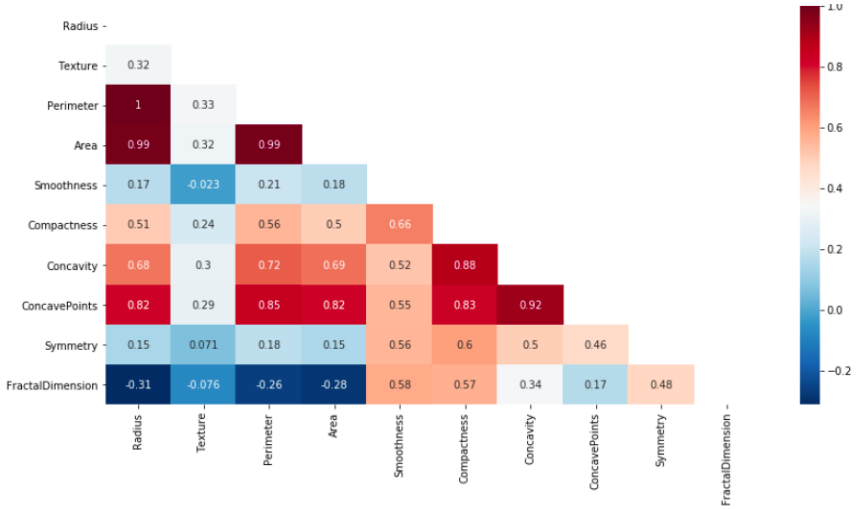
569 rows × 12 columns

Se agrupan los datos para saber cuántos tumores benignos y malignos fueron diagnosticados en los datos.

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

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

Se realiza la selección de variables utilizando un mapa de calor y las variables seleccionadas son: textura, área, smoothness, compactness, symmetry, FractalDimension y perímetro.




Se reemplaza la palabra *malignant* y *benignant* en el conjunto de datos para generar la clasificación.

```
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 |

Se seleccionan las variables que serán las predictoras y las variables a predecir para generar *dataFrames* a partir de ellas. Los conjuntos de datos generados son los siguientes.

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

#X = np.array(BCancer[['Radius', 'Texture', 'Perimeter',
#pd.DataFrame(X)
```

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 10.38 | 122.80 | 0.11840 | 0.27760 | 0.2419 | 0.07871 |
| 1 | 17.77 | 132.90 | 0.08474 | 0.07864 | 0.1812 | 0.05667 |
| 2 | 21.25 | 130.00 | 0.10960 | 0.15990 | 0.2069 | 0.05999 |
| 3 | 20.38 | 77.58 | 0.14250 | 0.28390 | 0.2597 | 0.09744 |
| 4 | 14.34 | 135.10 | 0.10030 | 0.13280 | 0.1809 | 0.05883 |
| ... | ... | ... | ... | ... | ... | ... |
| 564 | 22.39 | 142.00 | 0.11100 | 0.11590 | 0.1726 | 0.05623 |
| 565 | 28.25 | 131.20 | 0.09780 | 0.10340 | 0.1752 | 0.05533 |
| 566 | 28.08 | 108.30 | 0.08455 | 0.10230 | 0.1590 | 0.05648 |
| 567 | 29.33 | 140.10 | 0.11780 | 0.27700 | 0.2397 | 0.07016 |
| 568 | 24.54 | 47.92 | 0.05263 | 0.04362 | 0.1587 | 0.05884 |

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

| | 0 |
|---|---|
| 0 | 1001.0 |
| 1 | 1326.0 |
| 2 | 1203.0 |
| 3 | 386.1 |
| 4 | 1297.0 |
| ... | ... |
| 564 | 1479.0 |
| 565 | 1261.0 |
| 566 | 858.1 |
| 567 | 1265.0 |
| 568 | 181.0 |

569 rows × 1 columns

Se generó la división de datos para separar los datos de entrenamiento y de prueba.

```
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
                                                                    test_size = 0.2,
                                                                    random_state = 1234,
                                                                    shuffle = True)
```

```
pd.DataFrame(X_train)
#pd.DataFrame(X_test)
```

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 18.22 | 84.45 | 0.12180 | 0.16610 | 0.1709 | 0.07253 |
| 1 | 22.44 | 71.49 | 0.09566 | 0.08194 | 0.2030 | 0.06552 |
| 2 | 20.76 | 82.15 | 0.09933 | 0.12090 | 0.1735 | 0.07070 |
| 3 | 23.84 | 82.69 | 0.11220 | 0.12620 | 0.1905 | 0.06590 |
| 4 | 18.32 | 66.82 | 0.08142 | 0.04462 | 0.2372 | 0.05768 |
| ... | ... | ... | ... | ... | ... | ... |
| 450 | 15.18 | 88.99 | 0.09516 | 0.07688 | 0.2110 | 0.05853 |
| 451 | 15.10 | 141.30 | 0.10010 | 0.15150 | 0.1973 | 0.06183 |
| 452 | 18.60 | 81.09 | 0.09965 | 0.10580 | 0.1925 | 0.06373 |
| 453 | 18.70 | 120.30 | 0.11480 | 0.14850 | 0.2092 | 0.06310 |
| 454 | 13.78 | 81.78 | 0.09667 | 0.08393 | 0.1638 | 0.06100 |

Se generó el modelo de un bosque aleatorio de clasificación utilizando los datos de entrenamiento.

```
#Se entrena el modelo a partir de los datos de entrada
ClasificacionBA = RandomForestClassifier(random_state=0)
ClasificacionBA.fit(X_train, Y_train)

#ClasificacionBA = RandomForestClassifier(n_estimators=100, max_depth=8, min_samples_split=4, min_samples_leaf=2, ra
#ClasificacionBA.fit(X_train, Y_train)
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: DataConversionWarning: A column-vector y was pa
ssed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

Como en prácticas pasadas, utilizando los pronósticos del modelo inicial se genera el *score* a partir de los pronósticos y de los datos reales. El valor de .938 mejora respecto a la práctica donde solo se utilizaron árboles para la clasificación.

```
#Se calcula la exactitud promedio de la validación
ClasificacionBA.score(X_validation, Y_validation)
```

```
0.9385964912280702
```

Se genera una tabla de falsos positivos y falsos negativos y se obtienen buenos resultados para el modelo. Solo siete datos mal clasificados.

| Clasificación | Benign | Malignant |
|---|---|---|
| Real | | |
| **Benign** | 64 | 3 |
| **Malignant** | 4 | 43 |

También se generó un reporte de la clasificación. En dicho reporte se tiene el criterio que se utiliza, en este caso, gini. Además, se obtiene la importancia de cada una de las variables que se utilizaron, la precisión, el soporte entre otras. También se tiene la precisión de la predicción para cada una de las etiquetas.

```
#Reporte de la clasificación
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))
```

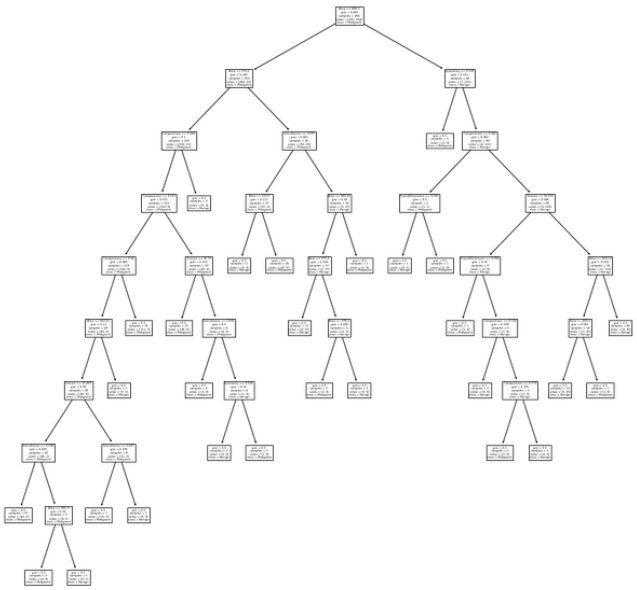
```
Criterio:
 gini
Importancia variables:
 [0.12681465 0.45987493 0.07943492 0.21421778 0.05473754 0.06492018]
Exactitud 0.9385964912280702
              precision    recall  f1-score   support

      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
```

Se generó la lista de la importancia de cada una de las variables seleccionadas para poder visualizar lo que cada una aporta al modelo.

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

Se graficó uno de los árboles que utiliza el bosque para poder observar su estructura y los niveles que contiene. El árbol graficado es el siguiente.



Además, se obtuvo la lista de cada una de las reglas que se obtienen utilizando texto para esta lista.

```
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
```

Como última acción para este modelo, se generaron dos predicciones utilizando el modelo generado para observar cómo funciona el modelo. Ambas predicciones obtuvieron el resultado correcto.

```
#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)
```

```
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)
```

```
array(['Benign'], dtype=object)
```

Por último se generó un modelo especificando los estimadores para poder observar el *score* que obtiene dicho modelo y si es una mejora al modelo que se obtuvo. Se creó el modelo utilizando 100 estimadores y se especificaron otros parámetros.

```
#Se entrena el modelo a partir de los datos de entrada
#ClasificacionBA = RandomForestClassifier(random_state=0)
#ClasificacionBA.fit(X_train, Y_train)

ClasificacionBA = RandomForestClassifier(n_estimators=100, max_depth=8, min_samples_split=4, min_samples_leaf=2, ran
ClasificacionBA.fit(X_train, Y_train)
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6: DataConversionWarning: A column-vector y was pa
ssed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=8, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=4,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

Se generan las clasificaciones

El *score* empeora por .01. Se tiene un modelo similar en *score* al generado con árboles. Este *score* puede mejorar o empeorar cambiando dichos parámetros, para ellos se tienen que buscar los que puedan generar el mejor modelo. Se muestra el *score* que se obtuvo.

```
#Se calcula la exactitud promedio de la validación
ClasificacionBA.score(X_validation, Y_validation)
```

```
0.9210526315789473
```

**Conclusiones**

Se utilizaron bosques aleatorios para generar una clasificación de maligno y benigno en tumores de pacientes con indicios de cáncer de mama. Utilizando este algoritmo se pudo generar un modelo capaz de predecir si un tumor es maligno o benigno a partir de ciertas características del tumor. Además, se compararon los resultados con los que se obtuvieron en prácticas pasadas utilizando un solo árbol para generar la clasificación. El *score* obtenido mejoró aunque por poco. Aún así, al igual que en los modelos de bosques aleatorios de pronóstico se puede concluir que se deben utilizar los bosques aleatorios antes que el árbol en solitario. Se tiene un menor sobreajuste y se tienen mejores modelos. Al igual que en la práctica pasada se obtuvo la estructura de uno de los árboles que conforman el bosque aleatorio. Por último, se generó otro bosque aleatorio utilizando diferentes parámetros para su generación y un número de 100 estimadores. Este modelo bajó en el *score* que se había conseguido, pero cambiando los parámetros y los estimadores se pueden obtener modelos con mejor *score* o que se ajusten a las necesidades del programador.