

25 DE MAYO DE 2023

PRÁCTICA 16

Clasificación Múltiple - Árboles de Decisión - Bosques Aleatorios



Objetivo de la práctica

Por Angel Damian Monroy Mendoza

No. de Cuenta: 316040707

Generar un modelo de clasificación múltiple de un conjunto de datos seleccionado por el alumno, con la finalidad de aplicar los algoritmos de machine learning: Árboles de Decisión y Bosques Aleatorios y comparar el desempeño que tienen los algoritmos.



Características:

- Existen dos conjuntos de datos están relacionados con variantes rojas y blancas del vino portugués "Vinho Verde". Hoy en día, el vino es disfrutado cada vez más por una gama más amplia de consumidores. El "Vinho Verde" de Portugal ha tenido crecimiento en sus exportaciones, por lo que se vuelve de interés el garantizar la calidad el mismo.
- A continuación, se muestra un diccionario de las variables relevantes para esta práctica

1. - fixed acidity (tartaric acid - g / dm³)
2. - volatile acidity (acetic acid - g / dm³)
3. - citric acid (g / dm³)
4. - residual sugar (g / dm³)
5. - chlorides (sodium chloride - g / dm³)
6. - free sulfur dioxide (mg / dm³)
7. - total sulfur dioxide (mg / dm³)
8. - density (g / cm³)
9. - pH
10. - sulphates (potassium sulphate - g / dm³)
11. - alcohol (% by volume)
12. - quality (score between 0 and 10)

Desarrollo

Para el desarrollo de esta práctica se realizaron varios procesos que se engloban en los siguientes puntos:

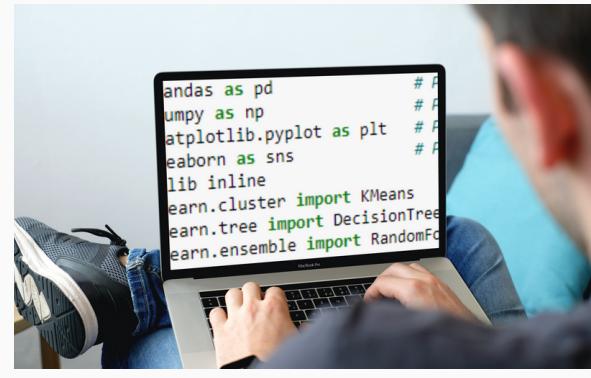
1. Importación de bibliotecas, acceso a datos y selección de características.
2. Modelo Híbrido: K - Means
3. Aplicación del algoritmo - Árboles de Decisión.
4. Aplicación del algoritmo - Bosques Aleatorios.
5. Comparación.

1. Bibliotecas, datos y selección de características

En este apartado se instalaron las bibliotecas necesarias para la manipulación de los datos, creación de vectores y matrices, generación de gráficas, la implementación de K - Means y, posteriormente, en la sección de *Aplicación del Algoritmo*, se instalaron las bibliotecas para los Árboles de Decisión y Bosques Aleatorios, así como lo necesario para saber la eficiencia del modelo y sus diferentes métricas de la matriz de confusión como *classification_report*, *confusion_matrix*, *accuracy_score* y *model_selection*.

Posteriormente, para la parte de cargar los datos se realizó mediante un link que se obtiene desde el archivo de GitHub, el cuál se instanció en la variable 'url' y se le pasó como argumento a la función de pandas para leer archivos con extensión csv. Después, se realizó un análisis exploratorio de datos muy superficial, en donde se observó la estructura de los mismos, si contaban con valores nulos o faltantes, una primera extracción de los datos totales de la variable objetivo: Calidad ('Quality'), una descripción estadística de los datos donde no se apreció valores atípicos aparentes y una impresión del mapa de calor de las correlaciones entre variables.

Para la selección de características se decidió reutilizar los resultados que se obtuvieron en la práctica de Análisis de Componentes Principales, puesto que fue el mismo DataSet, por lo que las variables que se eligieron se muestran en la siguiente página.



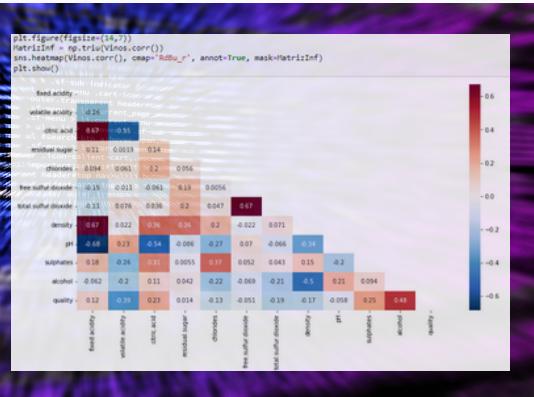
A screenshot of a Jupyter Notebook cell. The code imports pandas and reads a CSV file from a GitHub URL into a DataFrame named 'Vinos'. The notebook then displays the first few rows of the data and provides a statistical summary of the dataset.

```
url='https://raw.githubusercontent.com/aDamianMonroy/MiProyecto/main/WineQuality.csv'
Vinos = pd.read_csv(url, index_col = 'Unnamed: 0')
Vinos
```

	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides
1	7.4	0.700	0.00	1.9	0.07
2	7.8	0.880	0.00	2.6	0.09
3	7.8	0.760	0.04	2.3	0.09
4	11.2	0.280	0.56	1.9	0.07
5	7.4	0.700	0.00	1.9	0.07
...
1595	6.2	0.600	0.08	2.0	0.09
1596	5.9	0.550	0.10	2.2	0.06
1597	6.3	0.510	0.13	2.3	0.07
1598	5.9	0.645	0.12	2.0	0.07
1599	6.0	0.310	0.47	3.6	0.06

1599 rows × 12 columns

Desarrollo



Variables seleccionadas:

Debido a los resultados alcanzados en una práctica anterior, representativas eran:

- fixed.acidity.
- residual.sugar.
- chlorides.
- free.sulfur.dioxide.
- sulphates.
- alcohol.

2. K-Means

Para esta parte se utilizó la biblioteca `sklearn.preprocessing` para importar el método de normalizar y escalar, pues al estar trabajando con datos numéricos que difieren mucho entre ellos, es preciso realizar la estandarización para que cada uno de los valores tenga el mismo peso y aporte de forma equilibrada a los resultados.

Así pues, se eligió el método de estandarización para los datos (escalado) y, posteriormente, se instanció una nueva matriz estandarizada "MEstandarizada" ajustando los datos de la matriz obtenida de la selección de características, al escalado.

Luego, recordemos que con K-Means se tiene que proponer el número de clusters. Por esta razón, se utiliza el "método del codo" en donde se usa la suma total de la distancia al cuadrado dentro de cada grupo (SSE) y se realiza un recorrido entre 2 y 12 en donde se instancia el algoritmo `KMeans`, se ajusta este algoritmo a la matriz `MEstandarizada` y se realiza una lista sobre SSE con elementos de `km` con su atributo `.inertia_`. En seguida, se realiza una gráfica sobre estos SSE vs No. de Clústeres y, dado que en la práctica no siempre se tiene un punto de inflexión notable, se realiza un apoyo con el "método de la rodilla".

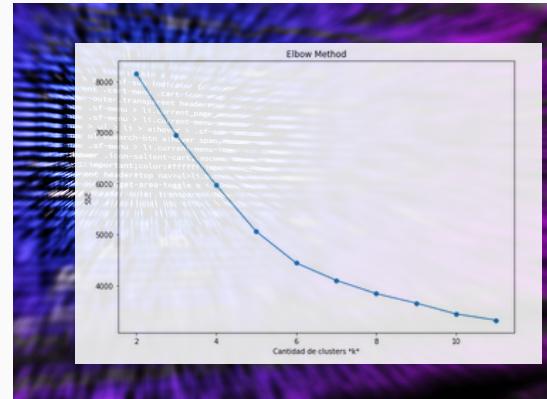




Desarrollo

Para el "método de la rodilla" se instaló kneed y, después, se importó la biblioteca de KneeLocator. Luego se instanció un localizador en el mismo rango de la SSE y la curva convexa y se imprimió el valor de **seis clusters** que arrojó el localizador.

Posteriormente, se crean las etiquetas de los elementos en los clústeres y se imprimen. Se agrega otra columna al conjunto de datos en donde se guardan los valores de las etiquetas para el clustering particional. Se hace una agrupación y conteo de los elementos en cada uno de los cluster y, por último, se obtienen los centroides para hacer el análisis de cada clúster.



3. Aplicación del algoritmo - Árboles de Decisión

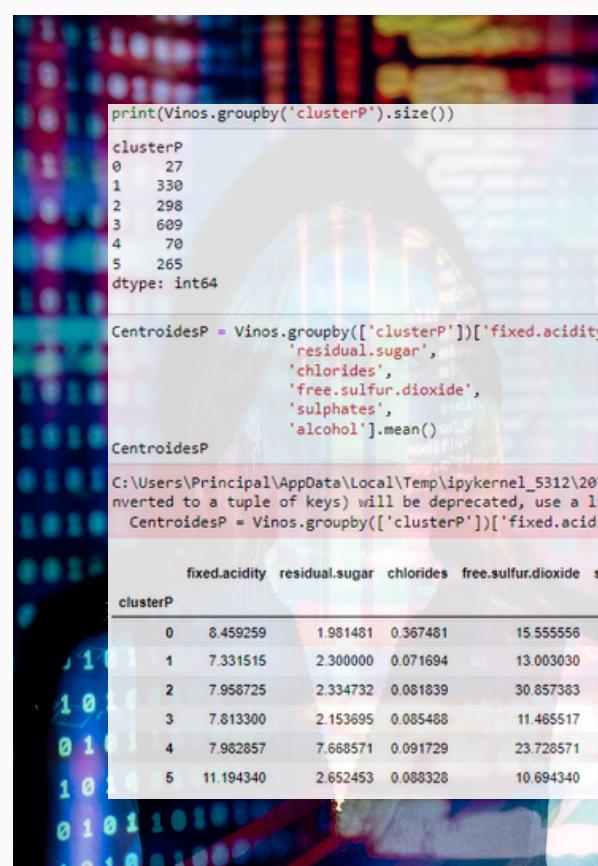
Para este apartado se instalaron las bibliotecas necesarias para la aplicación del algoritmo de árboles de decisión explicadas al principio de la práctica. Para ambos modelos se seleccionan las variables predictoras (X) y la variable a pronosticar (Y)

1er Modelo:

Variables predictoras: 'fixed.acidity', 'residual.sugar', 'chlorides', 'free.sulfur.dioxide', 'sulphates', 'alcohol'.

Variable a pronosticar: 'clusterP'

Cabe mencionar que, dado que fue un modelo mixto, la salida que se obtuvo del algoritmo de clusterización particional "K-Means", se convierte en la variable objetivo (entrada) de los algoritmos de Árboles y Bosques. En este entendido, para ambos modelos se hace la división de los datos utilizando la función de *model_selection* con el método *train_test_split*, en donde se le dan las variables predictoras y la variable a pronosticar; el tamaño del conjunto de testeo, el cual en este caso fue de 20% de prueba (p) y 80% de entrenamiento (e), recordando que es un parámetro que podemos modificar a nuestro criterio.



*Nota: no se presentaron los análisis de los clústeres debido a que éstas se encuentran en el cuaderno.

Desarrollo

3. Aplicación del algoritmo - Árboles de Decisión

Asimismo, se agrego un estado random para volver reproducible los mismos datos obtenidos todas las veces que corramos nuestro código y un `shuffle = 'True'`, que indica que los datos estén barajeados, es decir, que no agarre los primeros datos para el conjunto de entrenamiento y los últimos para el de prueba, sino que sea al azar la asignación de los datos de prueba y entrenamiento.

Posteriormente, se entrena el modelo a través de Árbol de Decisión Regresor; esto quiere decir que, creamos un objeto para instanciar la función `DecisionTreeRegressor()` junto con los hiperparámetros que vienen por defecto y luego, a ese objeto se le ajustan los datos de entrenamiento.

A continuación, con estos modelos ya entrenados se generan los pronósticos con el método `.predict` y las variables predictoras del testeo o prueba y se imprimen en un `DataFrame` de pandas.

Por último, se obtienen las características necesarias para poder armar nuestros modelos de árbol de decisión, así como para poder evaluar qué tan buenos son. En este sentido, se imprime la exactitud, la matriz de clasificación, la matriz de confusión y la importancia de las variables:

```
accuracy_score(Y_validation, Y_ClasicacionAD)
```

```
0.946875
```

Clasificación	0	1	2	3	4	5
Actual	0	4	0	0	1	0
0	64	0	4	0	1	
1	0	58	0	1	2	
2	0	1	0	123	0	4
3	0	0	0	8	1	
4	0	2	0	0	0	46
5	0	0	0	0	0	5

Variable	Importancia	
5	alcohol	0.282026
0	fixed.acidity	0.266954
3	free.sulfur.dioxide	0.231674
1	residual.sugar	0.102896
4	sulphates	0.083607
2	chlorides	0.032842

Criterio:
gini
Importancia variables:
[0.26695429 0.10289585 0.03284165 0.2316745 0.08360741 0.2820263]
Exactitud: 0.946875

	precision	recall	f1-score	support
0	1.00	0.80	0.89	5
1	0.96	0.93	0.94	69
2	1.00	0.95	0.97	61
3	0.96	0.96	0.96	128
4	0.89	0.89	0.89	9
5	0.85	0.96	0.90	48
accuracy			0.95	320
macro avg	0.94	0.91	0.93	320
weighted avg	0.95	0.95	0.95	320

*Nota: no se presentaron los análisis de las configuraciones debido a que éstas se encuentran en el cuaderno.

Desarrollo

4. Aplicación del algoritmo - Bosques Aleatorios

Para la aplicación del algoritmo de Bosque Aleatorio se realizó un proceso muy similar, pues también se crea un modelo para instanciar la función `RandomForestRegressor()` junto con los hiperparámetros que vienen por defecto y luego, a ese objeto se le ajustan los datos de entrenamiento. De igual forma se generan los pronósticos con el método `.predict` y las variables predictoras del testeo o prueba y se imprimen en un `DataFrame` de pandas.

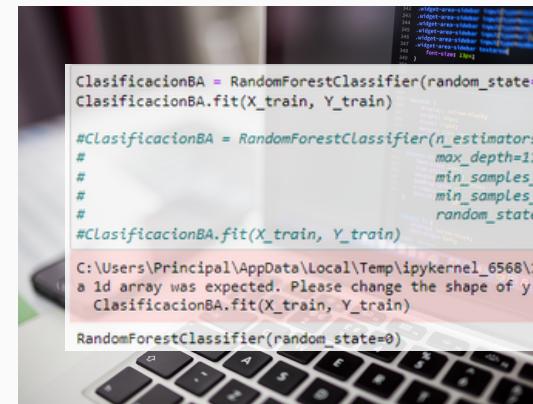
Asimismo, se obtienen las mismas métricas que se mencionan arriba de los árboles de decisión:

```
accuracy_score(Y_validation, Y_ClasicacionAD)  
0.946875
```

Clasificación	0	1	2	3	4	5
Reales						
0	4	0	0	1	0	0
1	0	67	0	2	0	0
2	0	0	57	2	1	1
3	0	0	0	127	0	1
4	0	0	0	0	9	0
5	0	2	0	0	0	46

```
Criterio:  
gini  
Importancia variables:  
[0.22741028 0.10566743 0.06378156 0.27000977 0.08851478 0.24461618]  
Exactitud: 0.96875  
precision recall f1-score support  
  
0 1.00 0.80 0.89 5  
1 0.97 0.97 0.97 69  
2 1.00 0.93 0.97 61  
3 0.96 0.99 0.98 128  
4 0.98 1.00 0.99 9  
5 0.96 0.96 0.96 48  
  
accuracy 0.97 0.97 0.97 320  
macro avg 0.97 0.94 0.95 320  
weighted avg 0.97 0.97 0.97 320
```

Por último, cabe recalcar que para cada modelo se obtuvo la conformación del Árbol, así como sus respectivas reglas. Cabe mencionar que para el caso de Bosques Aleatorios, no se puede generar directamente el árbol y las reglas, pues se tiene que elegir uno del total de 'estimadores' para poder imprimir estos objetos. Para ambos casos de Bosques Aleatorios se escogió el estimador 50.

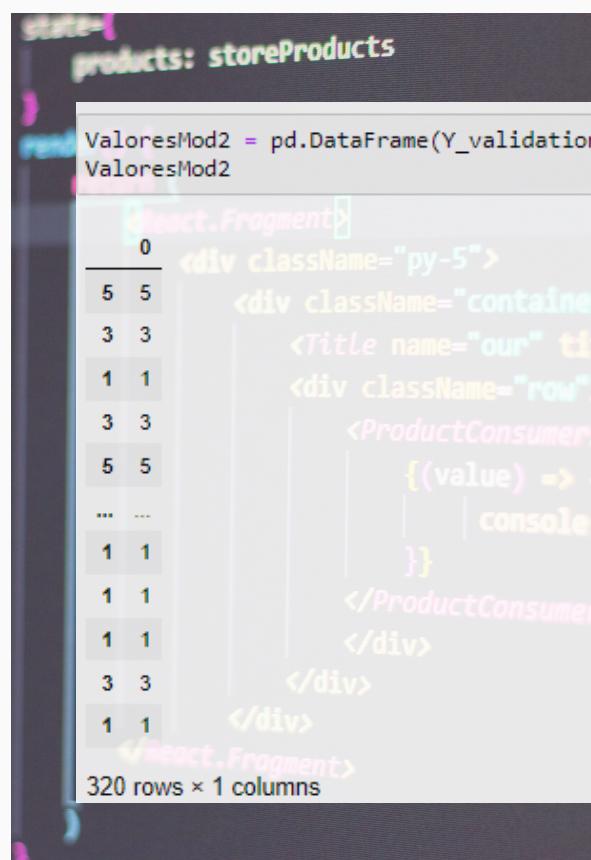


```
ClasificacionBA = RandomForestClassifier(random_state=0)
ClasificacionBA.fit(X_train, Y_train)

#ClasificacionBA = RandomForestClassifier(n_estimators=50,
#                                         max_depth=10,
#                                         min_samples_split=2,
#                                         min_samples_leaf=1,
#                                         random_state=0)

C:\Users\Principal\AppData\Local\Temp\ipykernel_6568\1035.py:11: UserWarning: n_estimators has been deprecated in version 0.22 in favor of estimator_n_estimators. It will be removed in 0.24.
  #ClasificacionBA.fit(X_train, Y_train)

RandomForestClassifier(random_state=0)
```



```
state=1  
products: storeProducts  
prod  
ValoresMod2 = pd.DataFrame(Y_validation)  
ValoresMod2  
  
<ProductFragment>  
0  
5 5  
3 3  
1 1  
3 3  
5 5  
...  
1 1  
1 1  
1 1  
3 3  
1 1  
<ProductFragment>  
320 rows × 1 columns
```

*Nota: no se presentaron los análisis de las configuraciones debido a que éstas se encuentran en el cuaderno.

Desarrollo

Nuevos pronósticos.

Por último, para el mejor modelo se realizó un pronóstico de un vino ficticio con ciertos valores de entrada (variables predictoras) y con el modelo que se guardó en el objeto de *ClasificacionBA*, se pronosticó la variable objetivo.

5. Comparación y Conclusiones

Además, se realizaron las validaciones de los modelos, observando la comparación de la exactitud y apreciando que, como de costumbre, el bosque aleatorio presenta mejor desempeño que el árbol de decisión.

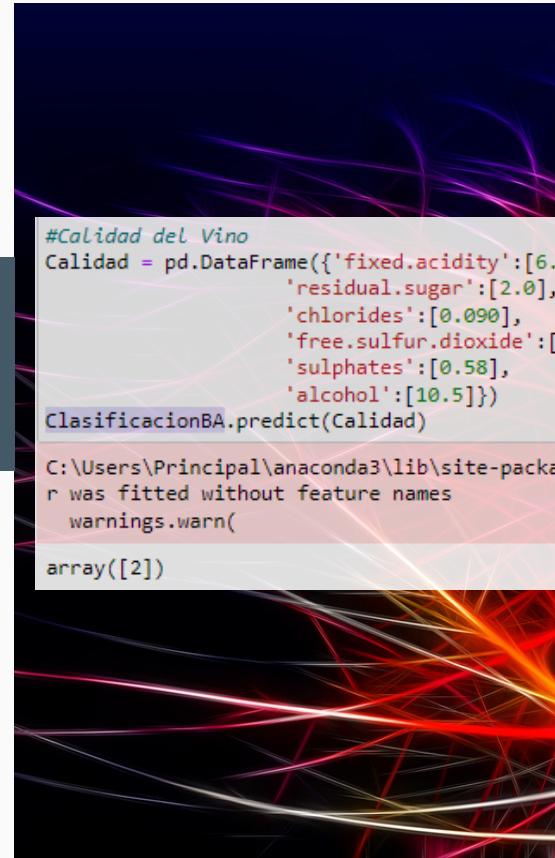
```
print("Árbol de decisión:", accuracy_score(Y_validation, Y_ClasificacionAD))
print("Bosque aleatorio:", accuracy_score(Y_validation, Y_ClasificacionBA))

Árbol de decisión: 0.946875
Bosque aleatorio: 0.96875
```

Por último, se analizó el rendimiento de cada una de las clases del modelo de bosque aleatorio obteniendo lo siguiente:

```
AUC para la clase 0: 0.9999999999999999
AUC para la clase 1: 0.9981234482360413
AUC para la clase 2: 0.9976897271979239
AUC para la clase 3: 0.9988810221354167
AUC para la clase 4: 0.997856377277599
AUC para la clase 5: 0.9978936887254902
```

De esta práctica podemos concluir que es muy importante la combinación de algoritmos no supervisados con supervisados, pues logramos mejorar nuestra clasificación múltiple del 68% (no registrado) a 96%. No obstante, estamos a expensas del especialista para catalogar las clases que se obtuvieron de la clusterización. Asimismo, observamos que durante dicha clusterización se encontraron 6 clases, pero con diferentes propiedades a las etiquetadas originalmente, tal y como se aprecia en las imágenes de a derecha, por lo que es interesante comprobar el nuevo etiquetado para saber realmente el significado



```
#Calidad del Vino
Calidad = pd.DataFrame({'fixed.acidity':[6.0, 7.0, 7.8, 10.0, 11.0, 12.0], 'residual.sugar':[2.0, 2.0, 2.0, 2.0, 2.0, 2.0], 'chlorides':[0.090, 0.090, 0.090, 0.090, 0.090, 0.090], 'free.sulfur.dioxide':[0.58, 0.58, 0.58, 0.58, 0.58, 0.58], 'sulphates':[0.58, 0.58, 0.58, 0.58, 0.58, 0.58], 'alcohol':[10.5, 10.5, 10.5, 10.5, 10.5, 10.5]})

ClasificacionBA.predict(Calidad)

C:\Users\Principal\anaconda3\lib\site-packages\sklearn\ensemble\bagged\_forest.py:217: UserWarning: Estimator not fitted, calling `predict` on it will result in an empty array
  "estimator not fitted, calling `predict` on it will result in an empty array", UserWarning)
array([2])
```

	fixed.acidity	residual.sugar	chlorides	free.sulfur.dioxide	sulphates	alcohol
quality						
3	8.360000	2.635000	0.122500	11.000000	0.570000	9.955000
4	7.779245	2.694340	0.090679	12.264151	0.596415	10.265094
5	8.167254	2.528655	0.092736	16.983847	0.620969	9.899706
6	8.347179	2.477194	0.084956	15.711599	0.675329	10.629519
7	8.872362	2.720603	0.076588	14.045226	0.741256	11.465913
8	8.566667	2.577778	0.068444	13.277778	0.767778	12.094444

	fixed.acidity	residual.sugar	chlorides	free.sulfur.dioxide	sulphates	alcohol
clusterP						
0	8.459259	1.981481	0.367481	15.555556	1.275926	9.470370
1	7.331515	2.300000	0.071694	13.003030	0.663091	11.836111
2	7.958725	2.334732	0.081839	30.857383	0.656174	10.129418
3	7.813300	2.153695	0.085488	11.465517	0.588834	9.758511
4	7.982857	7.668571	0.091729	23.728571	0.655429	10.425238
5	11.194340	2.652453	0.088328	10.694340	0.751283	10.168550