Cervantes Rubí Brandon

Inteligencia Artificial 2022-2, G.3, 22/05/2022
Practica No.16 Clasificación.

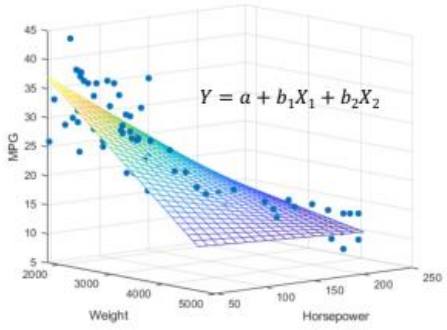
Objetivo.

Obtener una clasificación de gimnastas (De gimnasia artística varonil) que han participado en juegos olímpicos de acuerdo con sus características fisiológicas y regionales para saber si podrían o no ganar medalla.

Características.

Datos relacionados a deportistas olímpicos como su peso, altura y nacionalidad además las medallas que se obtuvieron durante sus competencias.

La regresión lineal múltiple es un algoritmo que busca calcular una salida Y (Una variable dependiente) en función de múltiples entradas (Variables dependientes) calculando la ecuación que genere la recta o el hiperplano el cual minimiza la distancia entre cada uno de los puntos y el hiperplano ajustado.



$$\hat{Y} = a + b_1x_1 + b_2x_2 + \cdots + b_nx_n + \mu$$

Donde μ, es el residuo del modelo que representa la diferencia entre el punto pronosticado y la observación real. Es importante considerar que al ser un algoritmo de aprendizaje supervisado los datos tienen que estar etiquetados, es decir, que todos deberán tener un valor real de la variable dependiente el cual se va a utilizar para entrenar al modelo.

Los árboles de decisión son algoritmos los cuales toman una serie de variables independientes como entrada y las analizan una por una en cada uno de los diferentes niveles del árbol partiendo de una raíz para obtener el determinado valor de una variable dependiente.

En este aspecto tendremos 2 diferentes tipos de árboles de decisión, primero los árboles de clasificación los cuales trabajan con etiquetas como resultados (A, B, C), (0/1), etc. y después están los árboles de regresión, dichos arboles están diseñados para entregar como resultado un valor continuo.

Los bosques aleatorios son algoritmos que buscan generalizar más las soluciones que nos pueden entregar los árboles aleatorios mediante la combinación de varios de estos árboles en la misma estructura de tal forma que cada uno de ellos aporte una solución similar para después llegar a un consenso evitando además un sobreajuste en los datos.

Al igual que en los árboles de decisión principalmente se trabajará con 2 tipos de estructuras para los bosques aleatorios, primero los bosques compuestos por árboles de clasificación los cuales trabajan con etiquetas como resultados (A, B, C), (0/1), etc. tomando aquella etiqueta que se repita más entre la salida de los árboles como la salida del modelo y después están los bosques compuestos por árboles de regresión, dichos arboles están diseñados para entregar como resultado un valor continuo y para los bosques se busca obtener un promedio con los resultados de cada árbol.

Cervantes Rubí Brandon

Inteligencia Artificial 2022-2, G.3, 22/05/2022
Practica No.16 Clasificación.

Desarrollo.

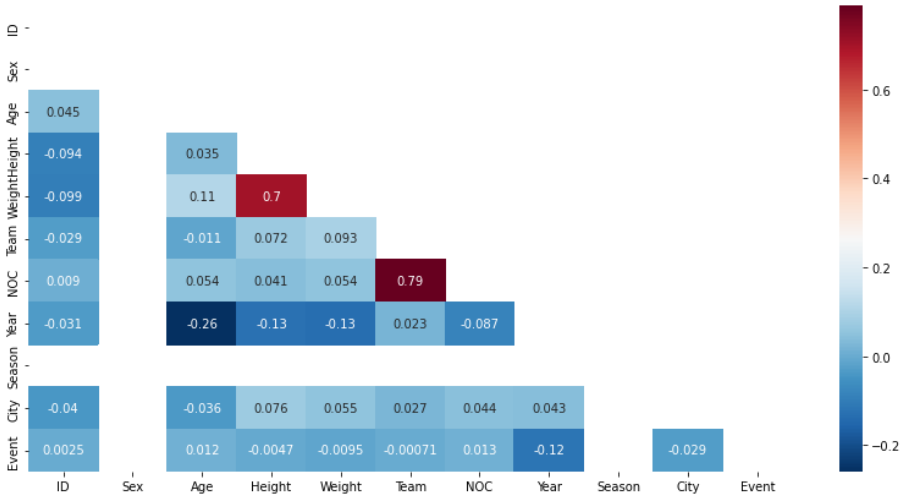
Como primer paso tenemos la importación de las librerías necesarias para trabajar con el conjunto de datos de los gimnastas a clasificar,s `pandas` para la manipulación de los datos, `numpy` para el manejo de matrices, `matplotlib` y `seaborn` para la visualización de estos datos mediante gráficos de diferente tipo dependiendo del análisis.

Después tenemos que hacer la lectura de nuestros datos los cuales se conforman de diferentes datos relacionados a diferentes cines como; el código de las películas proyectadas, los boletos vendidos, la ocupación de esa función, etc. todos estos datos se usaran en el entrenamiento para pronosticar el valor de ventas totales durante un determinado mes futuro.

Tenemos que leer los datos con ayuda de pandas el cual nos genera un `dataframe` con 271116 registros y 15 columnas asociadas a las características fisiológicas y regionales de los deportistas. El conjunto de datos leídos se muestra a continuación.

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
0	1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
1	2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
4	5	Christine Jacoba Aaftink	F	21.0	185.0	82.0	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	NaN

Para las 15 variables en el modelo se hizo un análisis de correlación de variables con el método de Pearson generando en primera instancia una matriz con valores de -1 a 1 siendo los más cercanos a los limites los valores que indican una relación fuerte entre las variables por lo cual se tendrán que quitar.



Del mapa de la página anterior, podemos determinar las variables predictoras finales del modelo eliminando el ID ya que es único para cada atleta, SEXO ya que es el mismo para todos los gimnastas (Masculino), NOC ya que presenta una relación fuerte con TEAM siendo ambos un indicativo de la región de donde proviene el atleta, de igual manera se elimina SEASON ya que todos los gimnastas de estas pruebas compiten en los juegos olímpicos de verano.

Al final del análisis de correlación las variables que se seleccionaron fueron las siguientes.

1) ID [SE ELIMINA]

2) Sex [SE ELIMINA]

3) Age [SE CONSERVA]*

4) Height [SE CONSERVA]*

5) Weight [SE CONSERVA]*

6) Team [SE CONSERVA]*

7) NOC [SE ELIMINA]

8) Year [SE CONSERVA]*

9) Season [SE ELIMINA]

10) City [SE CONSERVA] *

11) Event [SE CONSERVA]*

12) Name [SE ELIMINA]

13) Games [SE ELIMINA]

14) Medal [VARIABLE CLASE] *

15) Sport [SE ELIMINA]

Por lo cual nos quedaremos con 7 variables predictoras que nos ayudaran a encontrar el valor de la variable clase que será Medal para saber si un determinado deportista es prospecto para obtener una medalla, sea oro, plata o bronce.

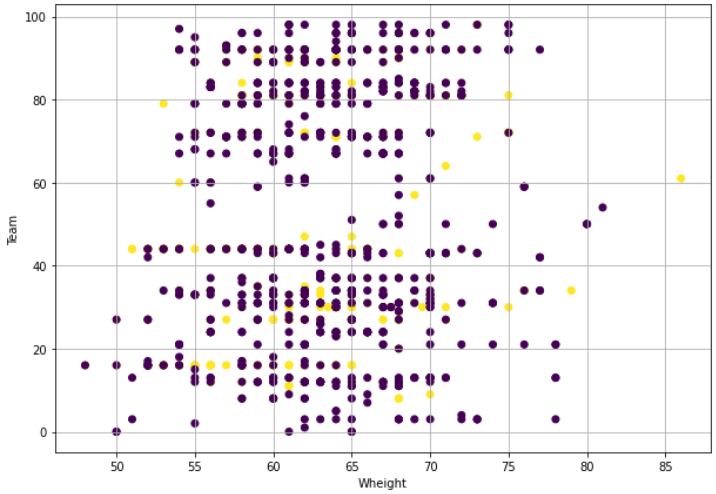
Con las variables seleccionadas, el siguiente paso fue filtrar a los deportistas que practiquen gimnasia artística varonil ya que el conjunto de datos engloba a muchos atletas. Para lograrlo primero se separó por gimnastas que practiquen la diciplina, después se quitaron todos los valores nulos quedando alrededor de 10, 000 registros. Para obtener un modelo que sea capaz de generalizar de manera correcta es pertinente equilibrar el número de registros que pertenecen a cada clase. De los 10, 000 deportistas alrededor de 650 obtuvieron medallas por lo cual se tomaron a todos estos y a 800 que no obtuvieron medalla para hacer la muestra de gimnastas del conjunto de datos.

```
#GIMNASTAS DE LA PRUEBA GENERAL ALL ARROUND
eventos = ["Gymnastics Men's Floor Exercise",
           "Gymnastics Men's Horizontal Bar",
           "Gymnastics Men's Horse Vault",
           "Gymnastics Men's Individual All-Around",
           "Gymnastics Men's Parallel Bars",
           "Gymnastics Men's Pommelled Horse",
           "Gymnastics Men's Rings",
           "Gymnastics Men's Team All-Around"]


#print(atletas[atletas['Sport'] == 'Gymnastics' ].groupby('Team').size())

gimnastas = atletas[atletas['Sport'] == 'Gymnastics' ]
gimnastas = gimnastas[gimnastas['Event'].isin(eventos)]

#Codificacion de variables nominales
enc = OrdinalEncoder()
gimnastas[["Team","Season", "City", "NOC", "Event", "Sex", "Season"]] =
enc.fit_transform(gimnastas[["Team","Season", "City", "NOC", "Event", "Sex",
"Season"]])

gimnastas
```

Primero se hizo un arreglo con todas las diciplinas y después se aplicó un filtro al conjunto de datos obtenido para hacer la separación, además se aplicó un codificador de sklearn llamado OrdinalEncoder para que las variables nominales del modelo (A, B, C, …) pudieran ser consideradas para el análisis ya que se consideraron importantes. Ejemplos de estas son: Team, Season, City y la variable clase Medal la cual se consideró como 1 para los que obtuvieron cualquier medalla y 0 para los que no.

```
medals_ = {'Gold': 1, 'Silver': 1, 'Bronze': 1}
gimnastas['Medal'] = gimnastas['Medal'].map(medals_)            # Cambiamos las medallas a
gimnastas['Medal'] = gimnastas['Medal'].fillna(0)               # Eliminamos los NaN de
gimnastas = gimnastas.dropna()                                  # Se eliminan todos los
medallas = gimnastas[gimnastas['Medal'] == 1.0 ]
SinMedalla = gimnastas[gimnastas['Medal'] == 0.0 ]
SinMedalla = SinMedalla.sample(n=800, random_state=1)
gimnastas = pd.concat([medallas, SinMedalla])
gimnastas
```

A continuación, se presenta una gráfica inicial con un comportamiento a simple vista no lineal.



El primer paso, que aplica para todos, consistió en separar el conjunto de datos en prueba (20% alrededor de 290 registros) y entrenamiento (80%, 1158 registros).

```
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y,
                                                                                test_size = 0.2,
                                                                                random_state = 1523,
                                                                                shuffle = True)
```

Regresión Logística.

Para generar el modelo se utilizó un objeto de la clase `linear_model.LogisticRegression` (Al cual llamamos Clasificacion) podemos utilizar el método `fit` pasando como datos los parámetros de los arreglos con los conjuntos de entrenamiento. Después de esto con `perdict_proba` y el conjunto de pruebas se crearon las probabilidades tanto para que sea 0 como para que sea 1 en cuanto a la clase, para redondear y tomar el valor más cercano a los limites se utilizó el modelo de clasificación generado a partir de estos datos con el método `predict` para generar las etiquetas predichas por el modelo y poder compararlas con las etiquetas reales para asignarle sus métricas de desempeño.

```
Clasificacion = linear_model.LogisticRegression()
Clasificacion.fit(X_train, Y_train)
……
Probabilidad = Clasificacion.predict_proba(X_validation)
pd.DataFrame(Probabilidad)

Predicciones = Clasificacion.predict(X_validation)
pd.DataFrame(Predicciones)
```

	0	1
0	0.669410	0.330590
1	0.758612	0.241388
2	0.784007	0.215993
3	0.387423	0.612577
4	0.859468	0.140532

La primera predicción (`predict_proba`) nos entregara un arreglo con las probabilidades para cada registro de que pertenezca a la etiqueta 0 y a la etiqueta 1 similar al de la derecha.

La segunda de ellas nos entrega un arreglo ordenado con las etiquetas predichas para cada uno de los registros en el conjunto de datos de prueba. Esto con el objetivo de asignar las métricas de desempeño, la primera de ellas que podemos obtener es el score con el método `score` de nuestro clasificador creado el cual tiene un valor de 93.85% que para el contexto medico con el que se está trabajando es un valor más que aceptable.

Podemos además generar la matriz de clasificación la cual nos muestra en la diagonal principal cual es el número de clasificaciones (positivas y negativas), todo lo que se encuentre fuera de esta diagonal se suma para contabilizar los errores. Podemos ver que el modelo se equivocó 105 veces de 290 posibles, cometiendo un número de errores similar de errores, sin embargo, cometiendo más errores en las variables en los valores clasificados como falsos negativos.

Clasificación	0.0	1.0
Real		
0.0	114	46
1.0	59	71

Adicional a esto también es posible generar una matriz la cual resuma el comportamiento de cómo es que se comporta el modelo de clasificación con los datos de prueba en cada una de las etiquetas.

	precision	recall	f1-score	support
0.0	0.66	0.71	0.68	160
1.0	0.61	0.55	0.57	130
accuracy			0.64	290
macro avg	0.63	0.63	0.63	290
weighted avg	0.64	0.64	0.64	290

En ella podemos ver que se comporta mejor clasificando aquellos gimnastas con posibilidades de medalla, el `recall` que indica la `sensibilidad` para el caso de las etiquetas positivas (1) y la `especificidad` para los casos negativos (0), entre otras métricas.

Con los atributos `coef_` e `intercept_` del modelo (Clasificación) podemos consultar cuales son los coeficientes de nuestro hiperplano el cual se va a ajustar a la mayoría de los registros siguiendo la siguiente ecuación.

$$\hat{Y} = \frac{1}{1 + e^{-(a+b_i x_i)}}$$

Donde $a + b_i x_i$ equivale a la ecuación de la regresión que se muestra a continuación.

$$a + bX = 0.00051486 - 0.01356838(Age) - 0.05737806(Height) - 0.00814632(Weight) + 0.00661377(Team) - 0.00424479(Year) - -0.01748977\ (City) + 0.24354815(Event)$$

Con este modelo se obtuvo una precisión general de 63.79% la cual es baja, sin embargo, ya se suponía un resultado similar debido a que el comportamiento de los datos es no lineal.

Arboles de decisión.

Con los conjuntos de datos establecidos, para generar el árbol, similar a la práctica anterior, es necesario utilizar la clase `DecisionTreeClassifier` para crear un objeto a partir de ella sobre la que es posible definir ciertos parámetros del árbol. Nos daremos cuenta de que los parámetros trabajados en esta ocasión son los mismos de la practica anterior, con `max_deep` podemos establecer la profundidad del árbol, con `min_samples_split` se define un mínimo de hojas que se tienen que generar y con `min_samples_leaf` podemos controlar el número de registros que definan una hoja de tal forma que los nodos hoja que no cumplan con este mínimo no se consideran para las reglas que se generen al final.

Seguido de esto fue necesario etiquetar de forma nominal los datos, y no discreta a diferencia de la regresión logística, para ello fue necesario transformar los 0 en la etiqueta 'medalla' y los 1 en 'sin medalla'.

```
medals_ = {1.0: 'Medalla', 0.0: 'SinMedalla'}
gimnastasArboles['Medal'] = gimnastasArboles['Medal'].map(medals_)
```

Cuando se crea el objeto con los parámetros establecidos, se usa el método `fit` pasando como parámetros los conjuntos de datos de entrenamiento que corresponden al 80% de los registros.

```
#MODIFICANDO LOS HIPERPARAMETROS
ClasificacionAD = DecisionTreeClassifier(max_depth=15, min_samples_split=4, min_samples_leaf=2, random_state=0)
ClasificacionAD.fit(X_train, Y_train)
```

Para generar las métricas de rendimiento del modelo primero es necesario utilizar el método `predict` pasando como parámetros el conjunto de datos de prueba que corresponde a las variables independientes del modelo `X_validation` para generar las clases que el modelo les asigne y comparar los resultados con el conjunto de datos que tiene los valores reales para estos registros, es decir `Y_validation`.

```
Y_Clasificacion = ClasificacionAD.predict(X_validation)
Valores = pd.DataFrame(Y_validation, Y_Clasificacion)
Valores
```

SinMedalla	SinMedalla
SinMedalla	SinMedalla
SinMedalla	Medalla
SinMedalla	SinMedalla
SinMedalla	SinMedalla

Con los valores de clase predichos por el modelo (Y_clasificacion a la izquierda) y los valores reales (Y_validation a la derecha) se creó un nuevo `dataframe` donde podemos hacer una comparación rápida entre ellos observando que en la mayoría de los casos los resultados de clasificación para los deportistas son correctos lo cual nos da un indicio de que el índice de precisión será mayor al obtenido con una regresión lineal.

Para validarlo se utilizó el método de nuestro objeto creado a partir de `DecisionTreeClassifier` llamado score y pasando como parámetros los conjuntos de variable clase y predictora de validación, es decir, el que corresponde al 20%.

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

Para considerar un punto de referencia primero se intentó hacer el pronóstico con un árbol al cual no se le modificaron los hiperparametros y se obtuvo un valor de precisión de 74.13%. Por otra parte, con una profundidad de 15 niveles, un mínimo de 4 registros por hoja para hacer la separacion y mínimo 2 registros para que se consideren nodos hoja se tiene una precisión de 72.41% en comparación con la regresión logística anterior donde se obtuvo un 63.79%. Al final se decidió que el modelo sin modificar los hiperparametros sería el adecuado para hacer las predicciones al tener el mayor índice de precisión sin caer en un sobreajuste como lo fue en el caso de la practica de algoritmos de pronóstico.

Para el modelo con la mayor precisión se generó su matriz de clasificación para poder observar a simple vista su comportamiento, en ella podemos a preciar todos los aciertos contabilizados en la diagonal principal y los errores como el complemento de esta diagonal principal.

```
#Matriz de clasificación
Y_Clasificacion = ClasificacionAD.predict(X_validation)
Matriz_Clasificacion = pd.crosstab(Y_validation.ravel(),
                                   Y_Clasificacion,
                                   rownames=['Real'],
                                   colnames=['Clasificación'])
Matriz_Clasificacion
```

Clasificación	Medalla	SinMedalla
Real		
Medalla	97	33
SinMedalla	42	118

Comparada con el modelo anterior podemos ver que, si bien se sigue equivocando, ya lo hace en menor medida, pero en esta ocasión equivocándose más en el caso de clasificar los calores falsos positivos, es decir que se predijo que se obtendría una medalla cuando en realidad no se gano ninguna medalla.

Se utilizo `classification_report` para obtener algunas de estas métricas, las cuales son las mismas que en el caso de la regresión logística; `recall` la cual debemos maximizar ya que es un índice que mide la habilidad del modelo para encontrar todas las clases relevantes en el data set, la `precisión` del modelo para asignar las clases y el `valor F1` cuyo objetivo es combinar la precisión y la exhaustividad (`recall`) en un solo indicé. Como todos estos índices varían según la configuración del árbol se detallarán debajo en el resumen de cada uno de los modelos.

```
#Reporte de la clasificación
print('Criterio: \n', ClasificacionAD.criterion)
print('Importancia variables: \n', ClasificacionAD.feature_importances_)
print("Exactitud", ClasificacionAD.score(X_validation, Y_validation))
print(classification_report(Y_validation, Y_Clasificacion))
```

	precision	recall	f1-score	support
Medalla	0.70	0.75	0.72	130
SinMedalla	0.78	0.74	0.76	160
accuracy			0.74	290
macro avg	0.74	0.74	0.74	290
weighted avg	0.74	0.74	0.74	290

Con el atributo `feature_importances_` podemos generar una lista ordenada con la importancia de cada una de las variables que se consideraron, a partir de esta lista se creó un dataframe ordenado por valor de importancia en el que podemos observar variaciones dependiendo de la configuración.

Para las tres configuraciones presentadas se generaron diferentes arboles partiendo de diferentes variables las cuales se detallan, pare generar las gráficas de los árboles y el conjunto de reglas fue necesario trabajar con los módulos `export_graphviz`, `plot_tree` y `export_text` de `sklearn.tree`.

Cervantes Rubí Brandon

Inteligencia Artificial 2022-2, G.3, 22/05/2022
Practica No.16 Clasificación.

Árbol generado.

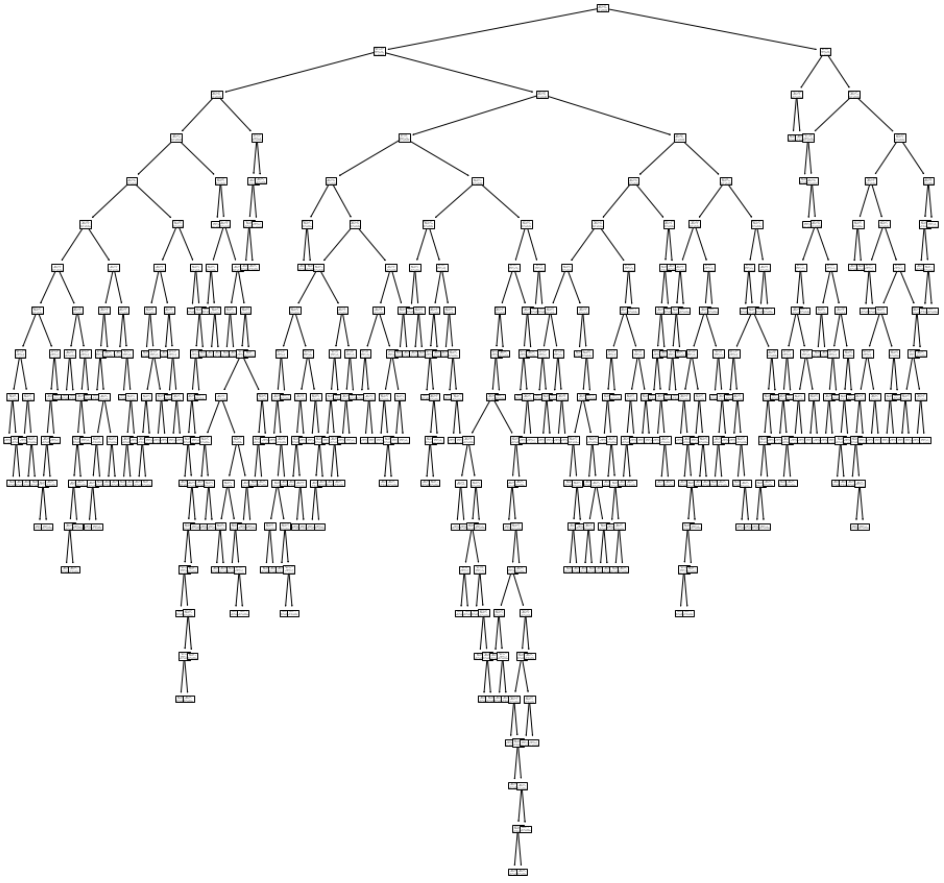
Nodo del que se parte.

Event <= 6.5
gini = 0.494
samples = 1158
value = [518, 640]
class = SinMedalla

	Variable	Importancia
6	Event	0.284838
3	Team	0.197601
2	Weight	0.114089
0	Age	0.105163
5	City	0.101057
4	Year	0.101053
1	Height	0.096199

El árbol generado presenta 21 niveles y el nodo raíz parte del tipo de evento con un score del 74.13%.

Evento es la variable que presenta un mayor nivel de entropía o que mayor importancia toma para el modelo, por el contrario, la menos importante es la altura.



Matriz de confusión del árbol.

Clasificación Real	Medalla	SinMedalla
Medalla	97	33
SinMedalla	42	118

Reporte de parámetros.

	precision	recall	f1-score	support
Medalla	0.70	0.75	0.72	130
SinMedalla	0.78	0.74	0.76	160
accuracy			0.74	290
macro avg	0.74	0.74	0.74	290
weighted avg	0.74	0.74	0.74	290

Por último, con `export_test` es posible generar una lista de reglas que definen la estructura de divisiones que va a tomar el árbol de decisión para hacer clasificaciones de nuevos registros, si se manda a imprimir podemos ver la estructura que toma el árbol desde otra perspectiva partiendo desde la variable más importante y haciendo las divisiones para las diferentes posibilidades del recorrido del árbol.

```
|--- Event <= 6.50
|   |--- Height <= 164.50
|   |   |--- Weight <= 62.50
|   |   |   |--- Team <= 51.50
|   |   |   |   |--- Team <= 33.50
|   |   |   |   |   |--- City <= 10.00
|   |   |   |   |   |   |--- City <= 4.50
|   |   |   |   |   |   |   |--- Weight <= 60.50
|   |   |   |   |   |   |   |   |--- Height <= 157.50
|   |   |   |   |   |   |   |   |   |--- Event <= 1.00
|   |   |   |   |   |   |   |   |   |   |--- class: Medalla
|   |   |   |   |   |   |   |   |   |--- Event >  1.00
|   |   |   |   |   |   |   |   |   |   |--- Event <= 2.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: SinMedalla
|   |   |   |   |   |   |   |   |   |   |--- Event >  2.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: Medalla
```

Bosques aleatorios.

Con las variables independientes del modelo seleccionadas se procedió con la aplicación del algoritmo, Para ello fue necesaria la clase `RandomForestClassifier, classification_report` de `sklearn`, y por parte del conjunto de entrenamiento y conjunto de pruebas se utilizaron los mismos del algoritmo de árboles de decisión.

Con los conjuntos de datos establecidos, para generar el bosque aleatorio fue necesario utilizar la clase `RandomForestClassifier` para crear un objeto a partir de ella sobre la que es posible definir ciertos parámetros de los árboles que conformaran el bosque, así como la cantidad. Algunos de los hiperparemertros más importantes son los mismos que en el caso de los árboles de decisión, con `max_deep` podemos establecer la profundidad del árbol, con `min_samples_split` se define un mínimo de hojas que se tienen que generar y con `min_samples_leaf` podemos controlar el número de registros que definan una hoja de tal forma que los nodos hoja que no cumplan con este mínimo no se consideran para las reglas que se generen al final y `n_estimators` el cual define el número de árboles o estimadores que participaran para solucionar el problema.

Para entrenar al modelo se usa el método `fit` del objeto de `RandomForestClassifier` creado pasando como parámetros los conjuntos de datos de entrenamiento que corresponden al 80% de los registros.

```
#MODIFICANDO LOS HIPERPARAMETROS
ClasificacionBA = RandomForestClassifier(n_estimators=50, max_depth=8,
min_samples_split=4, min_samples_leaf=2, random_state=0)
ClasificacionBA.fit(X_train, Y_train)
Y_Clasificacion = ClasificacionBA.predict(X_validation)
```

Con los valores de clase predichos por el modelo (`Y_clasificacion`) y los valores reales (`Y_validation`) se creó un nuevo `dataframe` donde podemos hacer una comparación rápida entre ellos observando que ahora el modelo se equivoca aun menos que con el algoritmo de regresión y aboles de decisión. Teniendo un mayor índice de errores en los valores falsos negativos, es decir que se clasificación como sin medalla, pero realmente ganaron una medalla.

Clasificación Real	Medalla	SinMedalla
Medalla	90	40
SinMedalla	22	138

Para obtener el índice de precisión del modelo se utilizó el método de nuestro objeto creado a partir de `RandomForestClassifier` llamado score y pasando como parámetros los conjuntos de variable clase y predictora de validación, es decir, el que corresponde al 20% alrededor de 200 registros.

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

En un principio se evaluó el modelo con los hiperparametros por defecto, es decir sin una limitante en la altura o los nodos hoja lo cual nos arrojó un modelo con un 78.27% de precisión el cual se encuentra por encima del obtenido por un solo árbol de decisión o la regresión logística variando solamente por décimas.

Se construyeron otras configuraciones modificando los hiperparametros relacionados al número de árboles que están en el bosque aleatorio a un total de 50, una altura de máximo 8 niveles en el árbol, con un mínimo de 2 registros en los nodos hojas y 4 registros en cada uno de ellos para hacer una nueva separación a modo de podar los que no cumplan con este requisito obteniendo un índice de precisión del 78.62% siendo el modelo que mejor clasifica los valores hasta ahora.

En esta ocasión también podemos obtener la matriz de clasificación y un reporte con las métricas descritas anteriormente para evaluar el desempeño del modelo. A continuación, se muestra el reporte con los valores de recall, F1, precisión y soporte de cada clase.

	precision	recall	f1-score	support
Medalla	0.80	0.69	0.74	130
SinMedalla	0.78	0.86	0.82	160
accuracy			0.79	290
macro avg	0.79	0.78	0.78	290
weighted avg	0.79	0.79	0.78	290

Con el atributo `feature_importances_` podemos generar una lista ordenada con la importancia de cada una de las variables que se consideraron, además se presenta la grafica de uno de los estimadores del bosque y su el conjunto de reglas. Fue necesario trabajar con los módulos `export_graphviz`, `plot_tree` y `export_text` de `sklearn.tree`.

Cervantes Rubí Brandon

Inteligencia Artificial 2022-2, G.3, 22/05/2022
Practica No.16 Clasificación.

Bosque aleatorio sin configurar los hiperparametros.

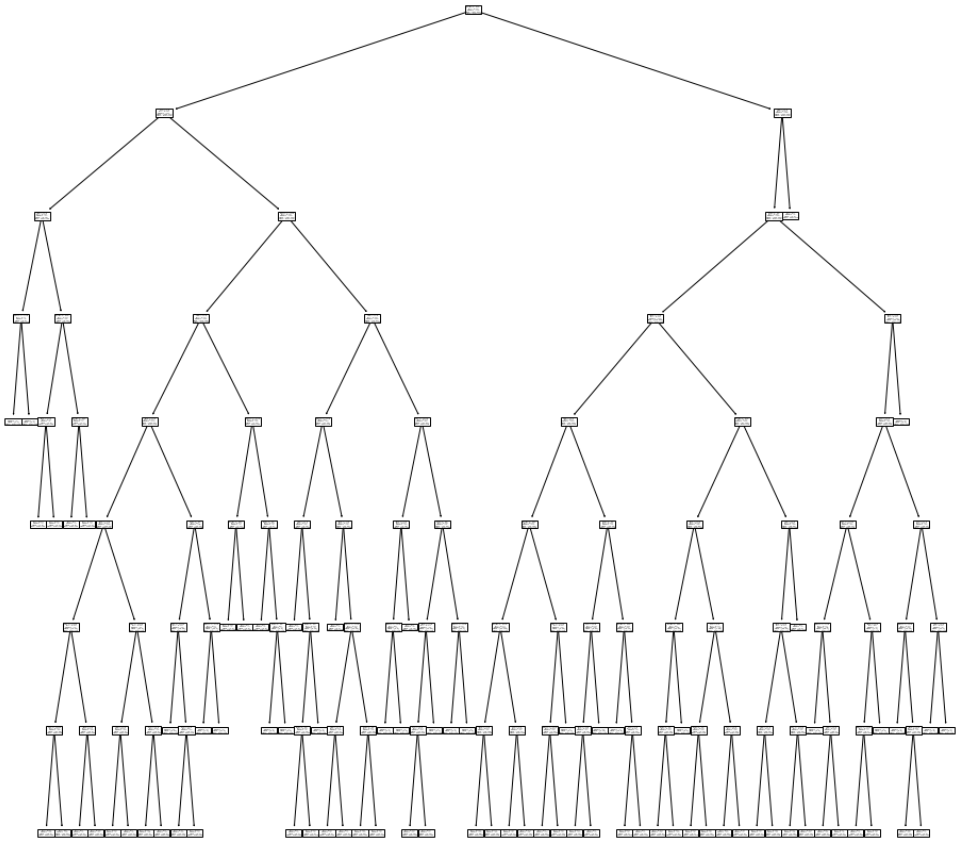
	Variable	Importancia
6	Event	0.296554
3	Team	0.218577
1	Height	0.128448
2	Weight	0.126916
4	Year	0.081066
0	Age	0.078995
5	City	0.069444

Nodo del que se parte.

Weight <= 61.5
gini = 0.497
samples = 736
value = [537, 621]
class = SinMedalla

El árbol generado presenta 8 niveles y el nodo raíz parte del peso del gimnasta con un score del 78.62%.

El peso es la variable que presenta un mayor nivel de entropía o que mayor importancia toma para el modelo, por el contrario, la menos importante es la ciudad donde se hicieron las competencias.



Matriz de confusión del árbol.

Clasificación Real	Medalla	SinMedalla
Medalla	90	40
SinMedalla	22	138

Reporte de parámetros.

	precision	recall	f1-score	support
Medalla	0.80	0.69	0.74	130
SinMedalla	0.78	0.86	0.82	160
accuracy			0.79	290
macro avg	0.79	0.78	0.78	290
weighted avg	0.79	0.79	0.78	290

Como ya se ha venido viendo podemos usar la librería `export_test` para generar una lista de reglas que definen la estructura de divisiones que va a tomar el árbol de decisión del bosque se seleccione para hacer clasificaciones con nuevos registros, si se manda a imprimir podemos ver la estructura que toma este árbol del bosque desde otra perspectiva partiendo desde la variable más importante dependiendo de la configuración, en la imagen se muestra el bosque con 50 árboles, una profundidad de 8, 4 registros en los nodos hoja y mínimo 2 nodos hoja de la página anterior.

```
|--- Weight <= 61.50
|   |--- City <= 2.50
|   |   |--- Age <= 22.50
|   |   |   |--- Height <= 163.50
|   |   |   |   |--- class: 0.0
|   |   |   |--- Height >  163.50
|   |   |   |   |--- class: 0.0
|   |   |--- Age >  22.50
|   |   |   |--- Weight <= 57.50
|   |   |   |   |--- Team <= 80.00
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- Team >  80.00
|   |   |   |   |   |--- class: 0.0
|   |   |   |--- Weight >  57.50
|   |   |   |   |--- Weight <= 59.50
|   |   |   |   |   |--- class: 1.0
```

Nuevos pronósticos.

Con los modelos obtenidos es posible hacer nuevas extrapolaciones ingresando nuevos valores para cada una de las variables independientes correspondientes a las características de los atletas que seleccionamos para el modelo en un inicio.

Para probar los tres modelos ingresamos datos de un gimnasta con una edad de 28 años, una altura de 175 cm, un peso de 64 kg, el Team 30 que corresponde a Finlandia, el año 1948, la ciudad 8 que corresponde a londres, y el evento con código 3, es decir all-arraund individual que es un promedio general de todas las pruebas. Se obtuvieron los siguientes resultados:

	Regresión Lineal Múltiple (63.79%)	Aboles aleatorios (74.13%)	Bosques Aleatorios (78.62%)
Valor pronosticado	Sin Medalla	Medalla	Medalla

Conclusiones.

En esta práctica se generaron tres modelos para la resolución de un problema de clasificación relacionado a si un gimnasta (De artística varonil) tenia posibilidades o no de ganar una medalla, independientemente de el tipo oro, plata o bronce. Primero se planteo un modelo de regresión logística el cual llego a tener un índice de precisión del 63.79%, bajo, pero ya se esperaba debido a que en la proyección de los datos no se observaba un comportamiento lineal claro. El segundo modelo fue con el algoritmo de bosques aleatorios para el cual se obtuvo un índice de precisión superior, de 74.13% sin embargo, este índice se mejoro con un bosque aleatorio al cual se le modificaron sus hiperparametros para obtener un índice de 78.62% que fue el mejor para el problema planteado.

En esta ocasión el sobreajuste no parece ser un problema que se presente ya que el indicé del score presenta un rango de incertidumbre de poco menos del 10% representando además un modelo que generaliza bien los casos para el contexto con el que se trabaja.

Fuentes.

pandas documentation — pandas 1.4.1 documentation. (2022). Pandas. 2022, de https://pandas.pydata.org/docs/index.html#

sklearn.tree.DecisionTreeRegressor. (2022). Scikit-Learn. 2022, de https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

sklearn.ensemble.RandomForestRegressor. (2022). Scikit-Learn. 2022, de https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

User guide: contents. (2022). Scikit-Learn. 2022, de https://scikit-learn.org/stable/user_guide.html

Dataset: https://www.kaggle.com/datasets/arashnic/cinema-ticket