

ANEXO

8 de septiembre de 2021

Practica 1: Clasificación usando árboles de decisión

Dataset: Pasajeros del Titanic <https://www.kaggle.com/c/titanic/data>

1. Objetivo de la práctica.

Construir un árbol de decisión para hacer clasificación de si un pasajero del Titanic sobrevive o no.

2. Conceptos.

Pandas es una librería de Python cuyo fin es la manipulación y análisis de datos, mediante el uso de Dataframes.

Pandas ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales de manera óptima.

Scikitlearn es una librería también escrita en Python, enfocada al aprendizaje automático. Incluye diferentes algoritmos de clasificación, regresión y análisis de grupos, entre ellos los árboles de decisión.

El dataset consiste en 3 archivos “.csv”: train (891), test (418), gender_submission (418), que contiene en total 1309 registros de pasajeros que viajaban en el Titanic en 1912.

3. Herramientas a usar.

Computadora con acceso a internet.
Los siguientes programas y librerías:

- a. Python versión 3.8.8

- b. Anaconda versión 4.10.1
- c. Jupyter-lab 3.0.14
- d. Pandas versión 1.2.4
- e. Scikitlearn versión 0.24.1
- f. Matplotlib versión 3.3.4

4. Desarrollo.

4.1. Entender el problema.

Se tiene un dataset donde se encuentra información de algunos de los pasajeros del Titanic. La información que más nos importa es saber si los pasajeros sobrevivieron o no. Para hacer esta clasificación de pasajeros que sobrevivieron o no, se propone usar un árbol de decisión, ya que este algoritmo es muy eficiente a la hora de clasificar los datos.

4.2. Criterio de evaluación.

Para este modelo, se propone usar accuracy (exactitud) como métrica de evaluación.

4.3. Preparar los datos.

- a. Descargue los datos del link que se encuentra al inicio de esta práctica.
- b. Extraer los datos.
Use el método mostrado en la práctica 0 sección 3 inciso h para extraer los datos con Python.

OPCIONAL

Cree una carpeta para extraer ahí los archivos.

- c. Cargar los datos.
Importar la librería Pandas.
Cargue los 3 archivos csv usando *read_csv()* y conviértalos a Dataframe ya que pandas tiene muchas herramientas para la manipulación de datos.

```
import pandas as pd
train = pd.DataFrame(pd.read_csv(Path+'train.csv'))
test = pd.DataFrame(pd.read_csv(Path+'test.csv'))
gen = pd.DataFrame(pd.read_csv(Path+'gender_submission.csv'))

```

Código 1

d. Explorar los datos.

Para eso Pandas tiene dos herramientas, la primera es `info()` y la otra es `head()`. La primera sirve para ver el nombre de las columnas y cuantos registros no vacíos tiene, el segundo muestra el contenido de las primeras 5 filas.

Debe poner atención en los registros vacíos ya que se debe hacer un preprocesamiento de los datos antes de trabajar con ellos. Use la herramienta `info()` para verificar si existen registros vacíos.

<pre><class 'pandas.core.frame.DataFrame'> RangeIndex: 891 entries, 0 to 890 Data columns (total 12 columns): # Column Non-Null Count Dtype --- --- 0 PassengerId 891 non-null int64 1 Survived 891 non-null int64 2 Pclass 891 non-null int64 3 Name 891 non-null object 4 Sex 891 non-null object 5 Age 714 non-null float64 6 SibSp 891 non-null int64 7 Parch 891 non-null int64 8 Ticket 891 non-null object 9 Fare 891 non-null float64 10 Cabin 204 non-null object 11 Embarked 889 non-null object dtypes: float64(2), int64(5), object(5) memory usage: 83.7+ KB</pre>					<pre><class 'pandas.core.frame.DataFrame'> RangeIndex: 418 entries, 0 to 417 Data columns (total 11 columns): # Column Non-Null Count Dtype --- --- 0 PassengerId 418 non-null int64 1 Pclass 418 non-null int64 2 Name 418 non-null object 3 Sex 418 non-null object 4 Age 332 non-null float64 5 SibSp 418 non-null int64 6 Parch 418 non-null int64 7 Ticket 418 non-null object 8 Fare 417 non-null float64 9 Cabin 91 non-null object 10 Embarked 418 non-null object dtypes: float64(2), int64(4), object(5) memory usage: 36.0+ KB</pre>				
--	--	--	--	--	---	--	--	--	--

(a) Train info

(b) Test info

Como se puede ver en train, hay valores faltantes en Age, Embarked y Cabin. Debido a que faltan demasiados datos en Cabin, es mejor quitar esa columna. PassengerId tampoco se usará debido a que solo es un número que ayuda a identificar los pasajeros.

En el caso de test, hay datos faltantes en Age, Fare y Cabin.

Usando la función `drop` elimine las columnas Cabin y PassengerId. [Ver figuras c y d]

```
train.drop(['Cabin', 'PassengerId'], axis = 1, inplace = True)
test.drop(['Cabin', 'PassengerId'], axis = 1, inplace = True)
```

e. Preprocesamiento de los datos.

Age

Una forma de rellenar los valores faltantes en Age es usar la mediana de los datos que ya se conocen. Llene los valores nulos con la mediana de los datos conocidos.

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S

(c) Train head

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	Q
1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	S
2	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	Q
3	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	S
4	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	S

(d) Test head

```
train['Age'].fillna(train['Age'].median(), inplace = True)
test['Age'].fillna(test['Age'].median(), inplace = True)
```

Código 2

Name

Similar al caso de la columna PassengerId, la columna Name tiene valores únicos cuyo objetivo es meramente para identificar, por lo tanto, no es un dato muy útil por sí solo.

El criterio que se debe tomar es si el sexo es F o M y si es mayor o no a 18 años (Basado en el protocolo de “Mujeres y niños primero”)

Use las columnas Sex y Age para crear una nueva columna llamada Alias, la cual debe contener si el pasajero es Mr, Mrs o Young Female/Male. Use el siguiente código para realizar el etiquetado.

```
def rename(age, sex):
    names = []
    for i, j in zip(age, sex):
        if i < 18:
            if j == "female":
                names.append("Young_Female")
            elif j == "male":
                names.append("Young_Male")
        else:
            if j == "female":
                names.append("Mrs")
```

```

        elif j == "male":
            .....names.append("Mr")
    return names

```

Código 3

Cree una nueva columna y asigne los los nuevos valores con el siguiente código.

```

train['Alias'] = train[['Age'], train[['Sex']]]
test['Alias'] = test[['Age'], test[['Sex']]]

```

Código 4

Verificar que la columna se ha creado usando la instrucción `head`. [Ver figuras e y f]

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Alias
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	Mr
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	Mrs
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	Mrs
3	1	1	Futelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	Mrs
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S	Mr

(e) Nueva columna train

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Alias
0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	Q	Mr
1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	S	Mrs
2	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	Q	Mr
3	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	S	Mr
4	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	S	Mrs

(f) Nueva columna test

Elimine las columnas Name y Ticket, ya que estas no dan información relevante.

```

train.drop(['Name', 'Ticket'], axis = 1, inplace = True)
test.drop(['Name', 'Ticket'], axis = 1, inplace = True)

```

Código 5

Embarked Usar la instrucción `mode()` en las columnas Embarked de `train` y `test`.

```
train['Embarked'].mode()
test['Embarked'].mode()
```

```
0    S
dtype: object
```

(g) Embarked train

(h) Embarked test

```
train['Embarked'] = train['Embarked'].fillna('S')
test['Embarked'] = test['Embarked'].fillna('S')
```

Código 6

Fare En el caso de Fare, al ser un solo dato faltante, podemos deducir el valor de este con base en la clase del pasajero, Inserte el siguiente código.[Ver figura i]

```
test[test['Fare'].isnull() == True]
```

Como ya se sabe que el dato faltante pertenece a la clase 3, es momento de ver cuál es la mediana de Fare en la clase 3. Para eso, use el siguiente código.[ver figura j]

```
test.groupby('Pclass').median()
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Alias
152	3	male	60.5	0	0	NaN	S	Mr

(i) Registro faltante

	Age	SibSp	Parch	Fare
Pclass				
1	39.0	0	0	60.0000
2	27.0	0	0	15.7500
3	27.0	0	0	7.8958

(j) Mediana de Fare

Llene el dato faltante con la mediana de `Pclass = 3`

```
test['Fare'] = test['Fare'].fillna(7.8958)
```

Código 7

Vuelva a revisar los datos usando `info()`. [Ver figuras k y l] Como se puede

#	Column	Non-Null Count	Dtype
0	Pclass	418 non-null	int64
1	Sex	418 non-null	object
2	Age	418 non-null	float64
3	SibSp	418 non-null	int64
4	Parch	418 non-null	int64
5	Fare	418 non-null	float64
6	Embarked	418 non-null	object
7	Alias	418 non-null	object

dtypes: float64(2), int64(3), object(3)
memory usage: 26.2+ KB

(k) Train info

#	Column	Non-Null Count	Dtype
0	Pclass	418 non-null	int64
1	Sex	418 non-null	object
2	Age	418 non-null	float64
3	SibSp	418 non-null	int64
4	Parch	418 non-null	int64
5	Fare	418 non-null	float64
6	Embarked	418 non-null	object
7	Alias	418 non-null	object

dtypes: float64(2), int64(3), object(3)
memory usage: 26.2+ KB

(l) Test info

ver, ya no hay registros vacios por lo tanto ya se puede empezar a trabajar con estos datos.

- f. Codificar los datos Esto se hace para convertir en números las variables categoricas, en este caso serían las columnas Sex, Embarked y Alias. Ingrese el siguiente código para codificar las columnas.

```
from sklearn.preprocessing import LabelEncoder
columnas = ['Sex', 'Embarked', 'Alias']
le = LabelEncoder()
#En el train
train_enc = train.iloc[:, :]
train_enc[columnas] = train_enc[columnas].apply(le.fit_transform)
#En el test
test_enc = test.iloc[:, :]
test_enc[columnas] = test_enc[columnas].apply(le.fit_transform)
#En el test
```

Código 8

Como se puede ver en las figuras m y n, ya solo se tienen valores numéricos en ambos datasets.

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Alias
0	3	1	34.5	0	0	7.8292	1	0
1	3	0	47.0	1	0	7.0000	2	1
2	2	1	62.0	0	0	9.6875	1	0
3	3	1	27.0	0	0	8.6625	2	0
4	3	0	22.0	1	1	12.2875	2	1

(m) Train codificado

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Alias
0	3	1	34.5	0	0	7.8292	1	0
1	3	0	47.0	1	0	7.0000	2	1
2	2	1	62.0	0	0	9.6875	1	0
3	3	1	27.0	0	0	8.6625	2	0
4	3	0	22.0	1	1	12.2875	2	1

(n) Test codificado

g. Dividir los datos.

Use este código para dividir los datos en X y y

```
y_train = train_enc['Survived']
X_train = train_enc.iloc[:,1:]
X_test = test_enc.iloc[:,:]

```

Código 9

Revise los nuevos datos usando `head` [Ver figuras ñ, o y p]

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Alias
0	3	1	22.0	1	0	7.2500	2	0
1	1	0	38.0	1	0	71.2833	0	1
2	3	0	26.0	0	0	7.9250	2	1
3	1	0	35.0	1	0	53.1000	2	1
4	3	1	35.0	0	0	8.0500	2	0

(ñ) X train

0	0
1	1
2	1
3	1
4	0

(o) y train

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Alias
0	3	1	34.5	0	0	7.8292	1	0
1	3	0	47.0	1	0	7.0000	2	1
2	2	1	62.0	0	0	9.6875	1	0
3	3	1	27.0	0	0	8.6625	2	0
4	3	0	22.0	1	1	12.2875	2	1

(p) X test

4.4. Construir el modelo.

Importe las siguientes librerías


```

from sklearn.tree import DecisionTreeClassifier , plot_tree
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score , confusion_matrix ,
plot_confusion_matrix , f1
import matplotlib.pyplot as plt

```

a. Construir el árbol.

Cree y entrene el árbol de decisión con los siguientes parámetros.

```

model = DecisionTreeClassifier(random_state = 0, criterion = 'gini',
max_depth = 5)
model.fit(X_train , y_train)

```

[Código 10](#)

Inserte este código para mostrar el árbol que se ha construido.

```

def arbol(tree , data):
    fig , ax = plt.subplots(figsize=(26, 12))

    print('Profundidad del arbol: ', tree.get_depth())
    print('Numero_de_nodos_terminales:', tree.get_n_leaves())

    plot = plot_tree(
        decision_tree = model,
        feature_names = data.columns.tolist(),
        class_names = 'Survived',
        filled = True,
        impurity = False,
        fontsize = 11,
        ax = ax)

```

[Código 11](#)

Llame a la función anterior con el modelo que recién construyó con el código 10

```

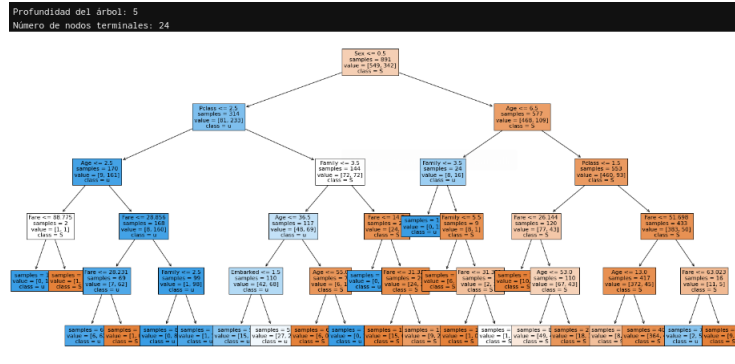
arbol(model , X_train)

```

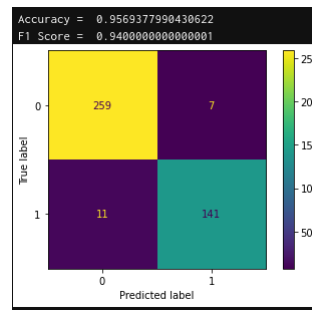
Debería mostrar lo siguiente. [Ver la figura q]

b. Predicciones y evaluación de X_test.

Use este código para realizar las predicciones e imprimir en pantalla los puntajes del modelo. [Ver figura r]



(q) Árbol



(r) Resultados

```
def predicciones(tree , X, label):
    y_pred = tree.predict(X_test)
    y_true = label["Survived"]
    plot_confusion_matrix(tree , X, y_true)
    print("Accuracy = ", accuracy_score(y_true , y_pred))
    print("F1 Score = ", f1_score(y_true , y_pred))

predicciones(model , X_test , gen)
```

Código 12

4.5. Análisis de errores.

Como se puede ver, el modelo tiene un score alto, lo que quiere decir que ha acertado al 96% de los datos pero esto se puede deber al sobre ajuste del modelo. Esto quiere decir que el modelo se ajustó demasiado a los datos con los que trabajó y esto puede hacer que falle en las predicciones de nuevos datos. Para disminuir este error Scikit-learn tiene una herramienta llamada Grid-SearchCV que mediante iteraciones y usando validación cruzada, trata de en-

contrar los mejores parámetros para un modelo.

Seleccione los parámetros a evaluar y construya el grid, usando un árbol vacío, los parámetros y con 10 validaciones cruzadas. Entrene ese grid con `X_train` y `y_train` [Ver figura s]

```

parametros = {'random_state': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'criterion': ['gini', 'entropy'], 'max_depth': range(1, 10),
              'min_samples_split': range(2, 10), 'min_samples_leaf': range(1, 5)}

grid = GridSearchCV(DecisionTreeClassifier(), parametros,
                    scoring='f1', cv=10)
grid.fit(X_train, y_train)

```

Código 13

Divida los datos de acuerdo al criterio

```

scores = pd.DataFrame(grid.cv_results_)
gini = scores[scores['param_criterion'] == 'gini']
entropy = scores[scores['param_criterion'] == 'entropy']

```

Código 14

Grafique el F1 promedio de acuerdo a “criterion” y “max_deep”. [Ver figura t]

```

plt.plot(gini['param_max_depth'], gini['mean_test_score'], 'o-')
plt.plot(entropy['param_max_depth'], entropy['mean_test_score'], 'o-')
plt.xlabel('param_max_depth')
plt.ylabel('mean_test_score')
plt.legend(['Gini', 'Entropy'])
plt.grid(True)

```

Código 15

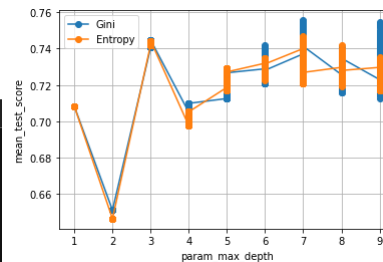
```

grid.fit(X_train, y_train)

GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                          'max_depth': range(1, 10),
                          'min_samples_leaf': range(1, 5),
                          'min_samples_split': range(2, 10),
                          'random_state': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
             scoring='f1')

```

(s) Entrenamiento grid



(t) Gini vs Entropy

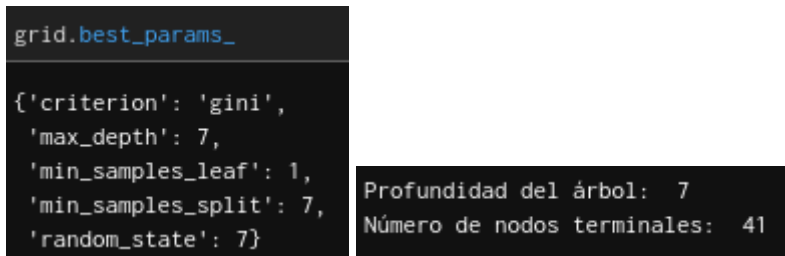
A primera vista se puede ver que usando gini y 8 niveles de profundidad se dan mejores resultados sin caer en el sobre ajuste.

Use `grid.best_params_` para ver los parámetros óptimos. [Ver figura u]

Use esos parámetros para construir un nuevo árbol de decisión. [ver figura v]

```
model_final = grid.best_estimator_
print('Profundidad del arbol: ', model_final.get_depth())
print('Numero de nodos terminales: ', model_final.get_n_leaves())
```

Código 16



```
grid.best_params_

{'criterion': 'gini',
 'max_depth': 7,
 'min_samples_leaf': 1,
 'min_samples_split': 7,
 'random_state': 7}

Profundidad del árbol: 7
Número de nodos terminales: 41
```

(u) Mejores parámetros

(v) Profundidad y nodos

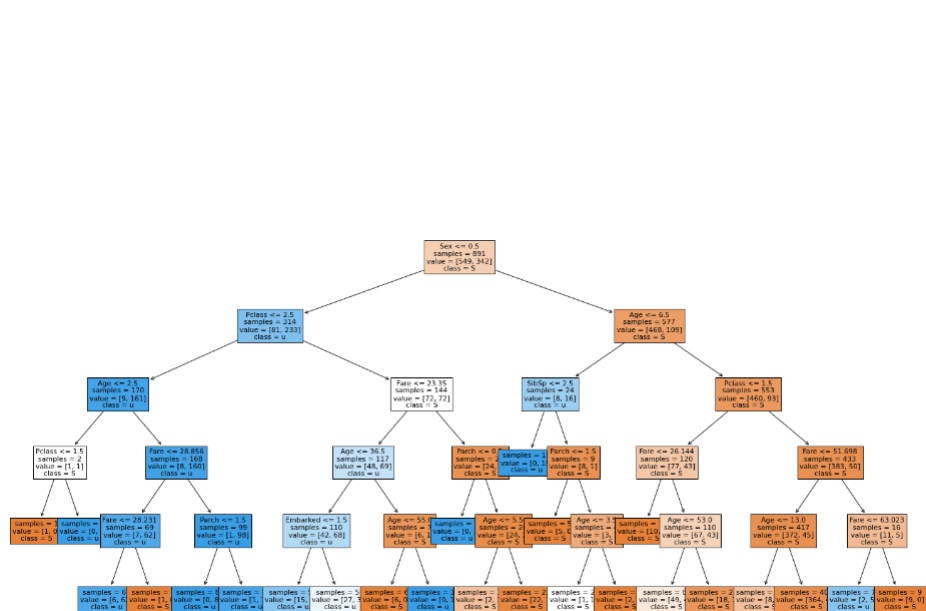
Use la función `arbol` en Código 11 para mostrar el nuevo árbol, recuerde cambiar `model` por `model_final`. [Ver figura w]

Y por último, realice nuevas predicciones con el nuevo modelo y evalúe el accuracy y f1 score usando la función del Código 12 usando el nuevo modelo. [Ver figura x]

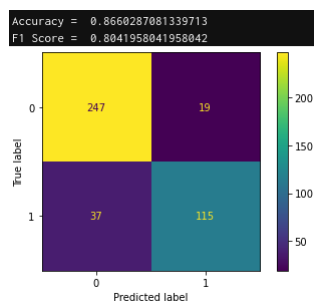
```
arbol(model_final, X_train)
predicciones(model_final, X_test, gen)
```

4.6. Implementación.

Ver el Notebook de Jupyter llamado *P1-Arbol-decision.ipynb*



(w) Nuevo árbol



(x) Nuevos Resultados