nxilf7lwl

September 12, 2023

#Momento de Retroalimentación: Módulo 2 Análisis y Reporte sobre el desempeño del modelo

##Naomi Padilla Mora A01745914

[1]:
```
from google.colab import drive

drive.mount("/content/gdrive")
!pwd  # show current path
```

```
Mounted at /content/gdrive
/content
```

#Librerias

[159]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

#Base de Datos

##Justificación

Se seleccionó la base de datos del Titanic debido a que cuenta con las dimensiones adecuadas para dividir los datos en Train, Test y validation. Además, esta base de datos fue previamente tratada y se realizó su limpieza en entregas posteriores. Por lo tanto, esta base de datos ya se encuentra limpia de valores nulos, duplicados y todos son numéricos.

Con esta base de datos, se obtuvo grandes resultados en el modelo antes implementado con Framework, mismo que se implementará en este trabajo pero ahora dividiendo los datos con train, test y validation.

Por último, el dataset cuenta con las variables suficientes para implementar un árbol de decisión y realizar la predección deseada sin la necesidad se analizar demasiadas variables, solo las elementales.

[72]:
```
# Cargar el conjunto de datos desde el archivo CSV descargado
df = pd.read_csv("/content/gdrive/MyDrive/concentración/árbol de decisión/
 ↪reporte/titanic.csv")
df
```

[72]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	1	22.0	1	0	7.2500	3
1	2	1	1	0	38.0	1	0	71.2833	1
2	3	1	3	0	26.0	0	0	7.9250	3
3	4	1	1	0	35.0	1	0	53.1000	3
4	5	0	3	1	35.0	0	0	8.0500	3
..	…	…	…	…	…	…	…	…	
884	887	0	2	1	27.0	0	0	13.0000	3
885	888	1	1	0	19.0	0	0	30.0000	3
886	889	0	3	0	28.0	1	2	23.4500	3
887	890	1	1	1	26.0	0	0	30.0000	1
888	891	0	3	1	32.0	0	0	7.7500	2

[889 rows x 9 columns]

#Limpieza del Dataset

##Información del dataset

Información básica sobre la base de datos.

- 9 columnas y 889 filas
- Inicialmente contamos con 9 variables numéricas (7 int y 2 float)

[73]:
```
#dimensiones del df
df.shape
```

[73]: (889, 9)

[74]:
```
#tipos de variables en el df
df.dtypes
```

[74]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Sex              int64
Age            float64
SibSp            int64
Parch            int64
Fare           float64
Embarked         int64
dtype: object
```

[75]:
```
#Comando para ver la cantidad de valores nulos en cada variable.
df.isna().sum()
```

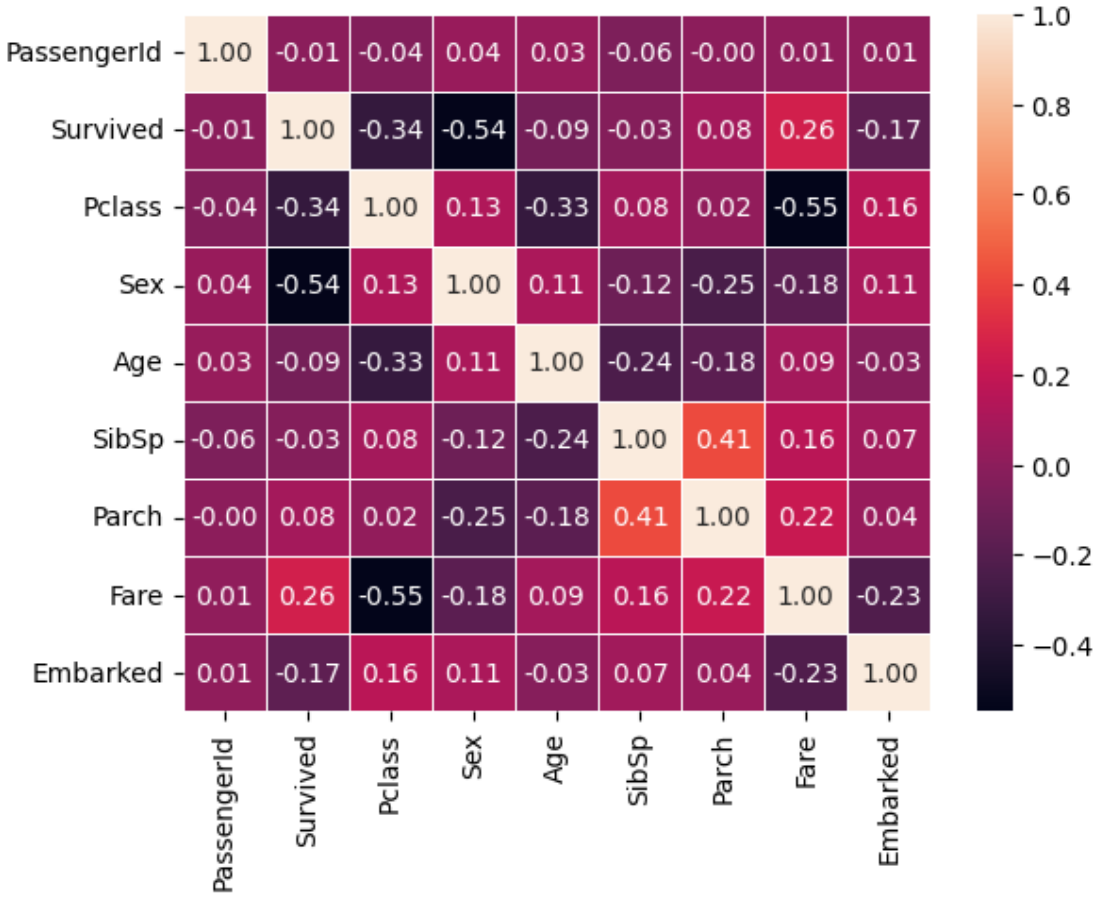
[75]:
```
PassengerId    0
Survived       0
Pclass         0
```

```
Sex            0
Age            0
SibSp          0
Parch          0
Fare           0
Embarked       0
dtype: int64
```

#Correlación

[76]:
```
sns.heatmap(df.corr(), annot=True, linewidth=.5, fmt=".2f")
```

[76]: <Axes: >



[77]:
```
cols = df.columns.to_numpy()
cols
```

[77]:
```
array(['PassengerId', 'Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
       'Parch', 'Fare', 'Embarked'], dtype=object)
```

Tomamos las variables con correlación > |0.15| respecto a la variable Survived que es la variable a predecir

[78]:
```
df = df[["Pclass","Sex","Fare","Embarked","Age","Survived"]]

df
```

[78]:

	Pclass	Sex	Fare	Embarked	Age	Survived
0	3	1	7.2500	3	22.0	0
1	1	0	71.2833	1	38.0	1
2	3	0	7.9250	3	26.0	1
3	1	0	53.1000	3	35.0	1
4	3	1	8.0500	3	35.0	0
..	…	…	…	…	…	…
884	2	1	13.0000	3	27.0	0
885	1	0	30.0000	3	19.0	1
886	3	0	23.4500	3	28.0	0
887	1	1	30.0000	1	26.0	1
888	3	1	7.7500	2	32.0	0

[889 rows x 6 columns]

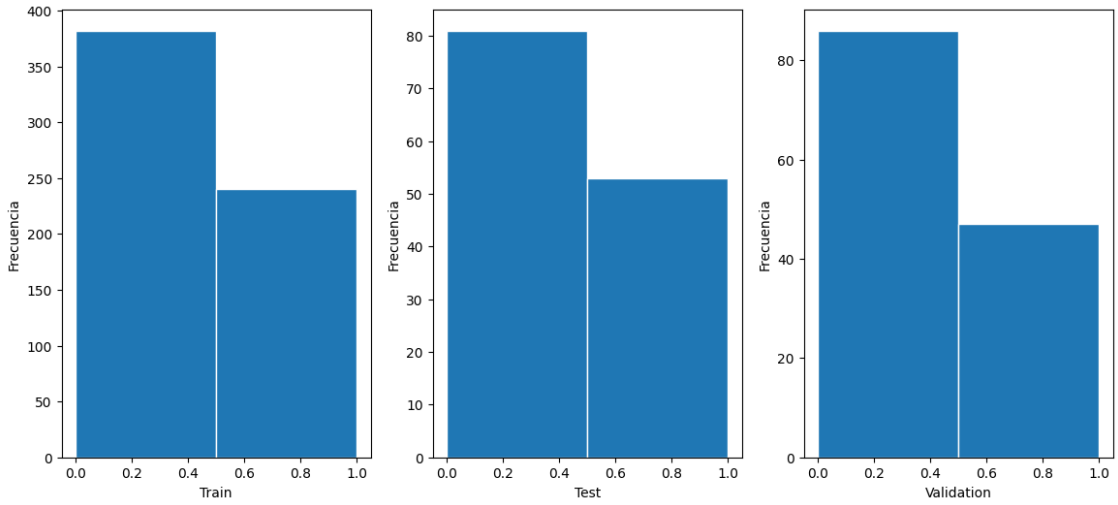
#Separación de los datos en train, test y validation

[153]:
```
X = df.drop("Survived", axis=1)  # Eliminar la columna de "Survived"
y = df["Survived"] #dejar la columna de "Survived"
```

[154]:
```
# Dividir los datos en conjunto de entrenamiento, validación y prueba
# Se divide en 70% entrenamiento, 15% validación y 15% prueba
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
 ↪random_state=42)
```

[167]:
```
plt.figure(figsize=(14, 6))
plt.subplot(131)
plt.hist(y_train, bins=[0, 0.5, 1], edgecolor = "white")
plt.xlabel('Train')
plt.ylabel('Frecuencia')
plt.subplot(132)
plt.hist(y_test, bins=[0, 0.5, 1], edgecolor = "white")
plt.xlabel('Test')
plt.ylabel('Frecuencia')
plt.subplot(133)
plt.hist(y_val, bins=[0, 0.5, 1], edgecolor = "white")
plt.xlabel('Validation')
plt.ylabel('Frecuencia')
```

[167]: Text(0, 0.5, 'Frecuencia')



#Árbol de decisión

[171]:
```
# Crear y entrenar un modelo de regresión de árbol de decisión
regressor = DecisionTreeRegressor(criterion='squared_error',␣
 ↪min_samples_split=2, max_depth=None)
regressor.fit(X_train, y_train)

# Hacer predicciones en el conjunto de validación
y_pred = regressor.predict(X_test)
```

[172]:
```
#solo correr
def metricas_modelo(y_test, predicciones):

#Métricas ideales para una regression

    # Mean Absolute Error (mae)
    # Prom. de las diferencias absolutas entre las predicc. y los valores␣
 ↪reales.
    # Cuanto menor sea el MAE, mejor será el rendimiento.
    mae = mean_absolute_error(y_test, y_pred)
    print("Mean Absolute Error (MAE):" ,mae)

    # Mean Square Error (mse)
    # Penaliza los errores más grandes al tomar la raíz cuadrada de la media de␣
 ↪los errores al cuadrado.
    # Igual que el mae, cuanto menor sea el MSE, mejor será el rendimiento.
    mse = mean_squared_error(y_test, y_pred)
    print("Mean Squared Error (MSE):", mse)
```

```
    # R-square (r**2)
    #Proporción de la varianza en los datos en el modelo.
    # Un valor cercano a 1 indica un buen ajuste del modelo.
    r2 = r2_score(y_test, y_pred)
    print("R-squared (R2):", r2)
```

[173]:
```
metricas = metricas_modelo(y_test, y_pred)
```

```
Mean Absolute Error (MAE): 0.2169509594882729
Mean Squared Error (MSE): 0.212039293329272
R-squared (R2): 0.11311960143945798
```