khqhtrhkt

September 14, 2023

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

1.1 Naomi Padilla Mora A01745914

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

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

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

2 Librerias

[2]:
```
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 DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
```

3 Base de Datos

3.1 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. Lo cual, permite la generalización del modelo, ya que así sabemos que este no se adapta a los datos.

Además, esta base de datos fue previamente tratada y se realizó su limpieza en entregas posteriores. Por lo tanto, ya se encuentra limpia de valores nulos, duplicados y todos son numéricos permitiendo tener un mayor enfoque en el desempeño del modelo.

Ya que la base de datos muestra a los pasajeros del titanic que sobrevivieron o no (0 = No, 1 = Yes), es perfecta para implementar un modelo de clasificación con árbol de decisión que hace esta

predicción binaria según los datos del pasajero como Clase, Sexo, Edad, etc.

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.

[3]:
```
# 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
```

[3]:

	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]

4 Limpieza del Dataset

4.1 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)

[4]:
```
#dimensiones del df
df.shape
```

[4]: (889, 9)

[5]:
```
#tipos de variables en el df
df.dtypes
```

[5]:
```
PassengerId      int64
Survived         int64
Pclass           int64
```

```
Sex              int64
Age            float64
SibSp            int64
Parch            int64
Fare           float64
Embarked         int64
dtype: object
```

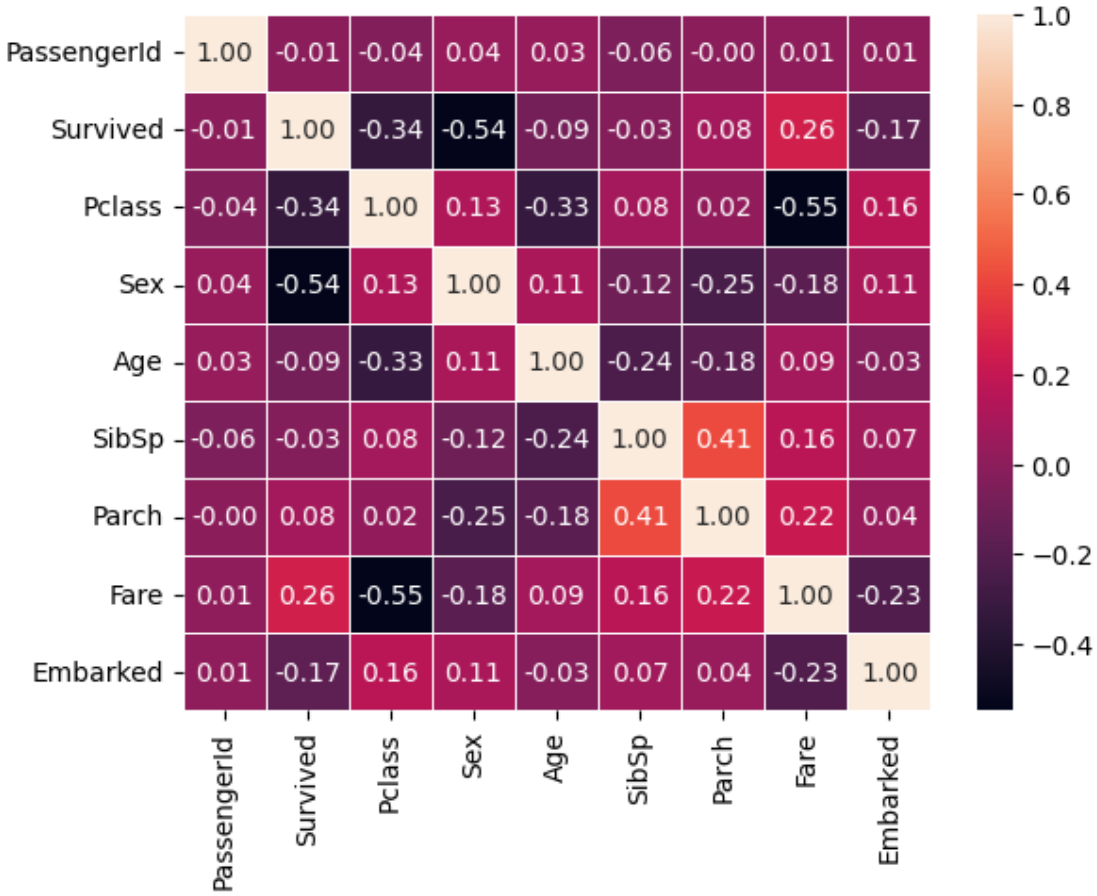
[6]:
```
#Comando para ver la cantidad de valores nulos en cada variable.
df.isna().sum()
```

[6]:
```
PassengerId    0
Survived       0
Pclass         0
Sex            0
Age            0
SibSp          0
Parch          0
Fare           0
Embarked       0
dtype: int64
```

4.2 Correlación

[7]:
```
sns.heatmap(df.corr(), annot=True, linewidth=.5, fmt=".2f")
```

[7]:
```
<Axes: >
```



[8]:
```
cols = df.columns.to_numpy()
cols
```

[8]:
```
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

[9]:
```
df = df[["Pclass","Sex","Fare","Embarked","Age","Survived"]]

df
```

[9]:

	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]
```

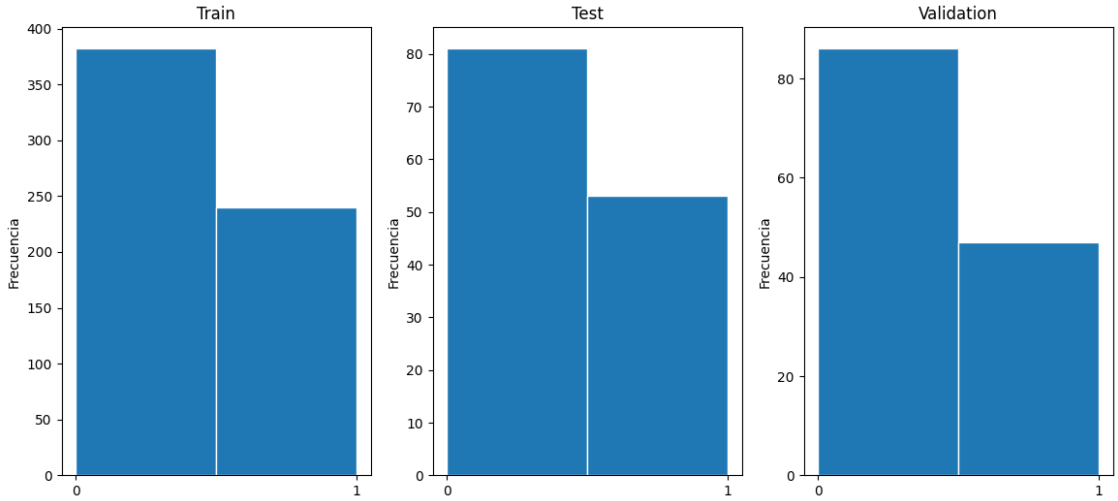
5 Separación de los datos en train, test y validation

[10]:
```
X = df.drop("Survived", axis=1)  # Eliminar la columna de "Survived"
y = df["Survived"] #dejar la columna de "Survived"
```

[11]:
```
# Dividir los datos en conjunto de entrenamiento, validación y prueba
# Se divide en 70% entrenamiento, 15% validación y 15% prueba
#Dividimos en train y test, siendo test el temp con un 30%
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
#Dividimos el temp en test y val con un 50% c/u
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
 ↪random_state=42)
```

[12]:
```
plt.figure(figsize=(14, 6))
plt.subplot(131)
plt.hist(y_train, bins=[0, 0.5, 1], edgecolor = "white")
plt.title('Train')
plt.ylabel('Frecuencia')
plt.xticks([0, 1], ['0', '1'])
plt.subplot(132)
plt.hist(y_test, bins=[0, 0.5, 1], edgecolor = "white")
plt.title('Test')
plt.ylabel('Frecuencia')
plt.xticks([0, 1], ['0', '1'])
plt.subplot(133)
plt.hist(y_val, bins=[0, 0.5, 1], edgecolor = "white")
plt.title('Validation')
plt.ylabel('Frecuencia')
plt.xticks([0, 1], ['0', '1'])
```

[12]:
```
([<matplotlib.axis.XTick at 0x79d66a86ddb0>,
  <matplotlib.axis.XTick at 0x79d66a86dd80>],
 [Text(0, 0, '0'), Text(1, 0, '1')])
```



Con los gráficos anteriores, podemos ver que el 70% de los datos se quedó en el train, y el 30% se ditribuye en un 50% en el Test y en el Validation. A continuación, se confirma esto mismo imprimiendo el shape de cada una de estas separaciones.

[13]:
```
print("X Train", X_train.shape, "X Test", X_test.shape, "X Val", X_val.shape)
```

```
X Train (622, 5) X Test (134, 5) X Val (133, 5)
```

[14]:
```
print("y Train", y_train.shape, "y Test", y_test.shape, "y Val", y_val.shape)
```

```
y Train (622,) y Test (134,) y Val (133,)
```

Nuestros conjuntos de datos son:

- Entrenamiento: X_train, y_train
- Prueba: X_test, y_test
- Validación: X_val, y_val

6 Implementación del modelo: Árbol de decisión

[15]:
```
# Crear y entrenar un modelo de árbol de decisión con una profundidad de 100
classifier = DecisionTreeClassifier(max_depth=100, random_state=42)

#Fit del modelo
classifier.fit(X_train, y_train)
```

[15]:
```
DecisionTreeClassifier(max_depth=100, random_state=42)
```

max_depth determina la profundidad de alcance del árbol, implicando la cantidad de niveles de decisión que puede tomar el árbol antes de dar una predicción final. A mayor sea este número,

implica mayor complejidad en el árbol, ya que en cada nivel puede obtener más detalle de las relaciones entre los datos de entrenamiento.

random_state es una semilla aleatoria para inicializar el modelo y su entrenamiento sea reproducible pero con los mismos resultados.

7 Predicciones conjunto de entrenamiento (Train)

[16]:
```
# Hacer predicciones en el conjunto de entrenamiento
y_pred = classifier.predict(X_train)
```

[17]:
```
# Evaluar el rendimiento del modelo
accuracy_train_o = accuracy_score(y_train, y_pred)
precision_train_o = precision_score(y_train, y_pred)
recall_train_o = recall_score(y_train, y_pred)
f1_train_o = f1_score(y_train, y_pred)

print(f'Accuracy Train: {accuracy_train_o}')
print(f'Precision Train: {precision_train_o}')
print(f'Recall Train: {recall_train_o}')
print(f'f1 Train: {f1_train_o}')

train = [accuracy_train_o, precision_train_o, recall_train_o, f1_train_o]
```

```
Accuracy Train: 0.9855305466237942
Precision Train: 0.9914893617021276
Recall Train: 0.9708333333333333
f1 Train: 0.9810526315789473
```

8 Predicciones conjunto de prueba (Test)

[18]:
```
# Hacer predicciones en el conjunto de prueba
y_pred = classifier.predict(X_test)
```

[19]:
```
# Evaluar el rendimiento del modelo
accuracy_test_o = accuracy_score(y_test, y_pred)
precision_test_o = precision_score(y_test, y_pred)
recall_test_o = recall_score(y_test, y_pred)
f1_test_o = f1_score(y_test, y_pred)

print(f'Accuracy Test: {accuracy_test_o}')
print(f'Precision Test: {precision_test_o}')
print(f'Recall Test: {recall_test_o}')
print(f'f1 Test: {f1_test_o}')

test = [accuracy_test_o, precision_test_o, recall_test_o, f1_test_o]
```

```
Accuracy Test: 0.7910447761194029
Precision Test: 0.7551020408163265
Recall Test: 0.6981132075471698
f1 Test: 0.7254901960784315
```

9 Predicciones conjunto de validación (Val)

[20]:
```
# Hacer predicciones en el conjunto de validación
y_pred = classifier.predict(X_val)
```

[21]:
```
# Evaluar el rendimiento del modelo
accuracy_val_o = accuracy_score(y_val, y_pred)
precision_val_o = precision_score(y_val, y_pred)
recall_val_o = recall_score(y_val, y_pred)
f1_val_o = f1_score(y_val, y_pred)

print(f'Accuracy Val: {accuracy_val_o}')
print(f'Precision Val: {precision_val_o}')
print(f'Recall Val: {recall_val_o}')
print(f'f1 Val: {f1_val_o}')

val = [accuracy_val_o, precision_val_o, recall_val_o, f1_val_o]
```

```
Accuracy Val: 0.7218045112781954
Precision Val: 0.5925925925925926
Recall Val: 0.6808510638297872
f1 Val: 0.6336633663366336
```

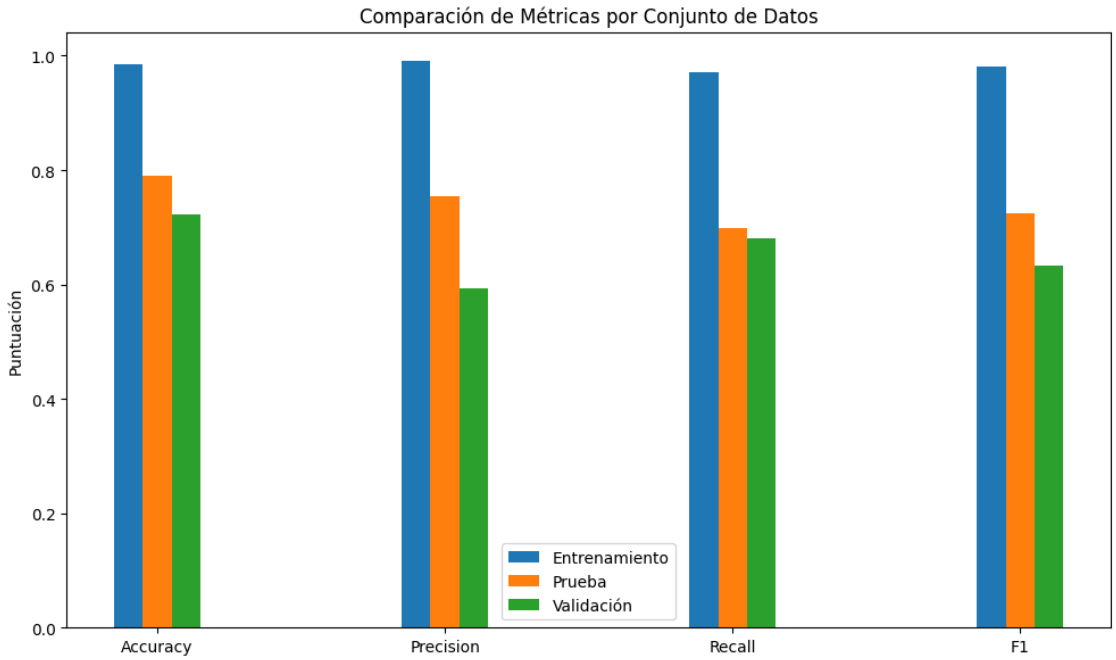
Inicialmente, viendo los puntajes obtenidos en las métricas de los tres conjuntos, podemos apreciar un sesgo en los datos de entrenamiento por la gran diferencia en los puntajes.

10 Diagnóstico y explicación el grado de bias o sesgo: bajo medio alto. Diagnóstico y explicación el nivel de ajuste del modelo: underfit, fit, overfit

[22]:
```
#gráfico comparativo de las métricas de precisión del modelo para cada conjunto␣
 ↪de datos
fig, ax = plt.subplots(figsize=(10, 6))
labels = ['Accuracy', 'Precision', 'Recall', 'F1']
x = np.arange(len(labels))
width = 0.1
rects1 = ax.bar(x - width, train, width, label='Entrenamiento')
rects2 = ax.bar(x, test, width, label='Prueba')
rects3 = ax.bar(x + width, val, width, label='Validación')

ax.set_ylabel('Puntuación')
```

```
ax.set_title('Comparación de Métricas por Conjunto de Datos')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()
plt.tight_layout()
plt.show()
```



Observando el gráfico, podemos confirmar el sesgo antes mencionado, el cual es alto. Es clara la diferencia de puntuación entre los valores obtenidos en las métricas de entrenamiento. Se observa una diferencia de más de 0.2 entre el train (barras azules) y test-val (barras naranjas y verdes). Esto implica que el modelo se ha ajustado demasiado bien a nuestros datos de entrenamiento, el cual es uno de los riesgos del decision tree. Ya que a mayor profundidad, también aumenta la probabilidad de sobreajuste implicando que en este modelo se observa un overfitting.

Este sesgo y sobreajuste también implica que el modelo no está generalizando o no le es posible generalizar propiamente, ya que las métricas en los res conjuntos deberían de ser más cercanas entre sí.

11 Diagnóstico y explicación el grado de varianza: bajo medio alto

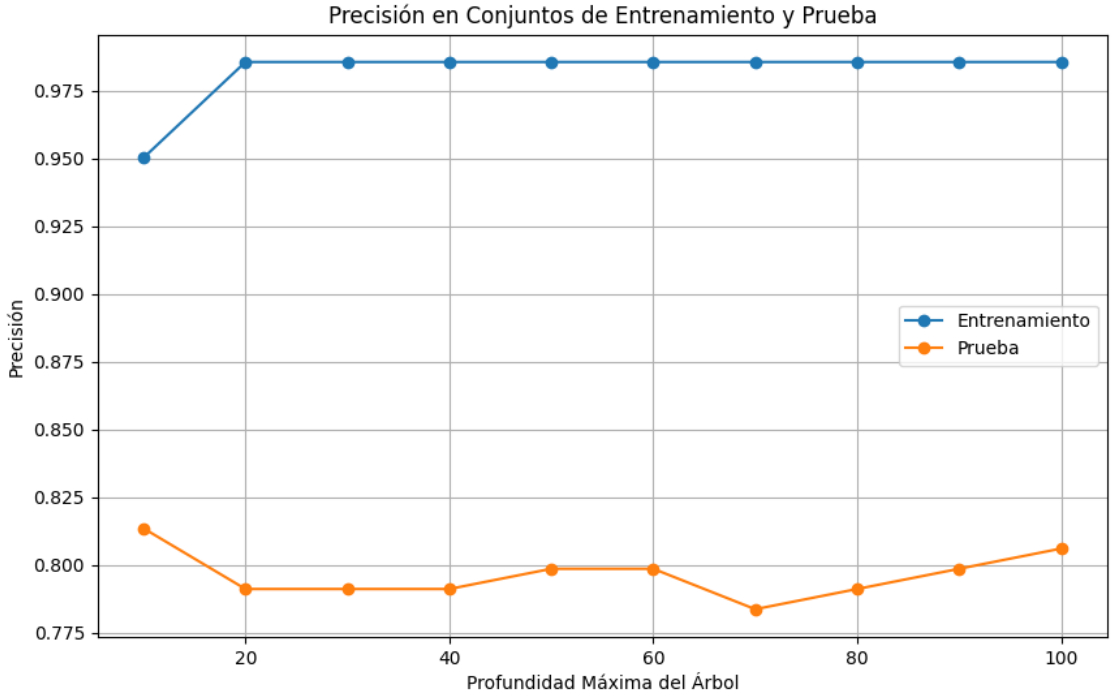
[23]:
```
param_values = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

train_accuracy = []
test_accuracy = []
```

```
# Itera a través de los valores del parámetro y entrena un modelo para cada uno
for param in param_values:
    # Crea un modelo de árbol de decisión con la profundidad máxima especificada
    clf = DecisionTreeClassifier(max_depth=param)
    clf.fit(X_train, y_train)

    # Calcula la precisión en el conjunto de entrenamiento y prueba
    train_accuracy.append(clf.score(X_train, y_train))
    test_accuracy.append(clf.score(X_test, y_test))

plt.figure(figsize=(10, 6))
plt.plot(param_values, train_accuracy, marker='o', label='Entrenamiento')
plt.plot(param_values, test_accuracy, marker='o', label='Prueba')
plt.title('Precisión en Conjuntos de Entrenamiento y Prueba')
plt.xlabel('Profundidad Máxima del Árbol')
plt.ylabel('Precisión')
plt.legend()
plt.grid(True)
plt.show()
```



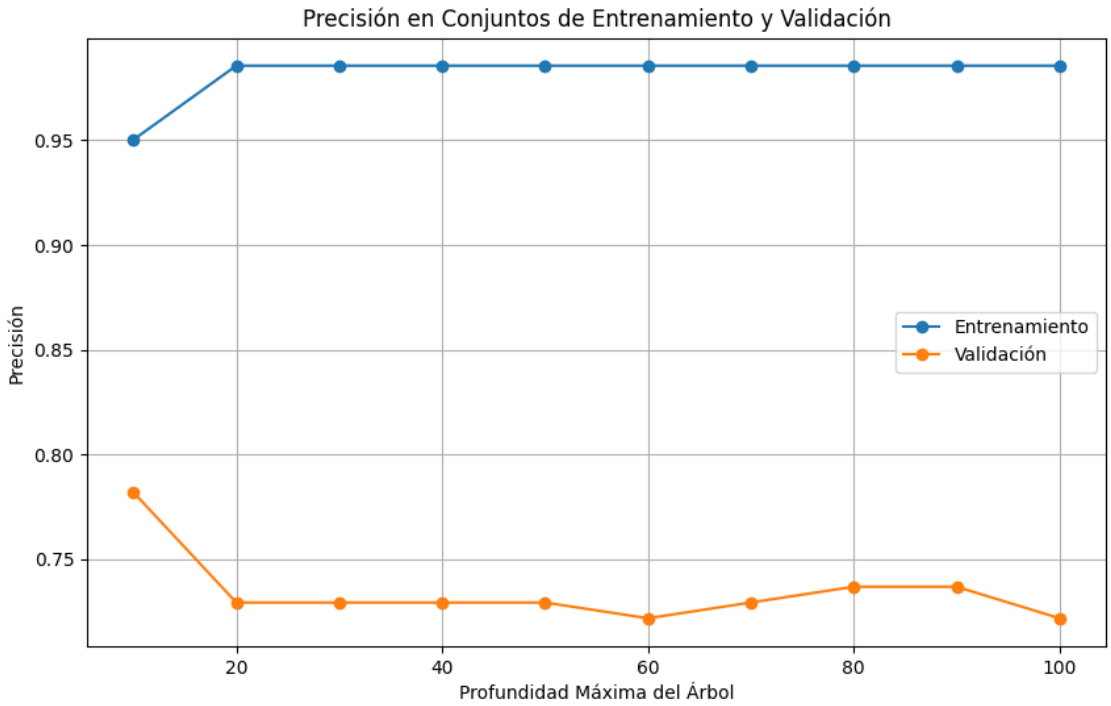
[24]:
```
param_values = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

train_accuracy = []
val_accuracy = []
```

```python
# Itera a través de los valores del parámetro y entrena un modelo para cada uno
for param in param_values:
    # Crea un modelo de árbol de decisión con la profundidad máxima especificada
    clf = DecisionTreeClassifier(max_depth=param)
    clf.fit(X_train, y_train)

    # Calcula la precisión en el conjunto de entrenamiento y prueba
    train_accuracy.append(clf.score(X_train, y_train))
    val_accuracy.append(clf.score(X_val, y_val))

plt.figure(figsize=(10, 6))
plt.plot(param_values, train_accuracy, marker='o', label='Entrenamiento')
plt.plot(param_values, val_accuracy, marker='o', label='Validación')
plt.title('Precisión en Conjuntos de Entrenamiento y Validación')
plt.xlabel('Profundidad Máxima del Árbol')
plt.ylabel('Precisión')
plt.legend()
plt.grid(True)
plt.show()
```



Haciendo la comparativa de estos gráficos podemos apreciar que el grado de varianza entre el train-test y el train-val es bastante significativo. Esto reafirma el sesgo y sobreajuste que se presenta en el modelo entendiento que este realmente solo esta sobreaprendiendo el comportamiento en los

datos de entrenamiento a mayor profundidad.

12 Uso de técnicas de regularización o ajuste de parámetros para mejorar el desempeño

12.1 Técnica 1: GridSearchCV

Se utilizará la función GridSearchCV la cual nos ayudará a buscar la mejor combiandción de parámetros posibles para el decision tree con el objetivo de mejorar el modelo.

- max_depth: profundidad máxima del árbol
- min_samples_split: cantidad min de muestras a dividir en un nodo
- min_samples_leaf: cantidad min de muestras en la hoja
- criterion: función de impureza a utilizar (gini, entropy)

[25]:
```
from sklearn.model_selection import GridSearchCV

# Define un diccionario de hiperparámetros y sus valores posibles
param_grid = {
    'max_depth': [5, 10, 15, None],  # Profundidad máxima del árbol
    'min_samples_split': [2, 5, 10],  # Mínimo de muestras para dividir un nodo
    'min_samples_leaf': [1, 2, 4],  # Mínimo de muestras en una hoja
    'criterion': ['gini', 'entropy'],  # Función de impureza
}

# Crea el modelo de DecisionTreeClassifier solo con random_state
classifier = DecisionTreeClassifier(random_state=42)

# Crea un objeto GridSearchCV
grid_search = GridSearchCV(classifier, param_grid, cv=5, scoring='accuracy')

# Realiza la búsqueda de hiperparámetros en los datos
grid_search.fit(X_train, y_train)

# Imprime los mejores hiperparámetros encontrados
print("Mejores hiperparámetros:", grid_search.best_params_)

# Imprime la mejor puntuación (precisión) encontrada durante la búsqueda
print("Mejor puntuación:", grid_search.best_score_)

# Obtén el modelo con los mejores hiperparámetros
best_classifier = grid_search.best_estimator_

# Evalúa el modelo con los mejores hiperparámetros en el conjunto de prueba
accuracy = best_classifier.score(X_test, y_test)
print("Precisión en el conjunto de prueba con mejores hiperparámetros:",␣
 ↪accuracy)
```

```
Mejores hiperparámetros: {'criterion': 'entropy', 'max_depth': 10,
'min_samples_leaf': 2, 'min_samples_split': 10}
Mejor puntuación: 0.8215870967741935
Precisión en el conjunto de prueba con mejores hiperparámetros:
0.8432835820895522
```

Como podemos ver, la mejor combinación de hiperparámetros que podemos realizar de nuestro modelo es la siguiente:

- **criterion: entropy**. La función de cálculo de impureza.
- **max_depth: 10**. La profundidad máx del árbol, si obervamos los gráficos previos podemos ver que es en una produndidad de 10 donde la varianza entre el train-test o train-val es menor.
- **min_samples_leaf: 2**. Se dividirá un min de 2 muestras por nodo.
- **min_samples_split: 10**. Se dividirá un min de 10 muestras por hoja.

Ahora evaluaremos este modelo en cada uno de los conjuntos de datos (train,test y val) para realizar una comparativa.

[26]:
```
# Crear y entrenar un modelo de árbol de decisión con los hiperparámetros de␣
↪GridSearch
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=10,␣
↪random_state=42, min_samples_leaf=2, min_samples_split=10)

#Fit del modelo
classifier.fit(X_train, y_train)
```

[26]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_leaf=2,
                       min_samples_split=10, random_state=42)
```

[28]:
```
# Hacer predicciones en el conjunto de entrenamiento
y_pred_train = classifier.predict(X_train)

# Hacer predicciones en el conjunto de validación
y_pred_test = classifier.predict(X_test)

# Hacer predicciones en el conjunto de validación
y_pred_val = classifier.predict(X_val)

# Evaluar el rendimiento del modelo (train)
accuracy_train_gs = accuracy_score(y_train, y_pred_train)
precision_train_gs = precision_score(y_train, y_pred_train)
recall_train_gs = recall_score(y_train, y_pred_train)
f1_train_gs = f1_score(y_train, y_pred_train)
print(f'Resultados del modelo con los hiperparámetros del GridSearch')
print(f'\n')
print(f'Accuracy Train: {accuracy_train_gs}')
print(f'Precision Train: {precision_train_gs}')
print(f'Recall Train: {recall_train_gs}')
print(f'f1 Train: {f1_train_gs}')
```

```python
# Evaluar el rendimiento del modelo (test)
accuracy_test_gs = accuracy_score(y_test, y_pred_test)
precision_test_gs = precision_score(y_test, y_pred_test)
recall_test_gs = recall_score(y_test, y_pred_test)
f1_test_gs = f1_score(y_test, y_pred_test)
print(f'\n')
print(f'Accuracy Test: {accuracy_test_gs}')
print(f'Precision Test: {precision_test_gs}')
print(f'Recall Test: {recall_test_gs}')
print(f'f1 Test: {f1_test_gs}')

# Evaluar el rendimiento del modelo (val)
accuracy_val_gs = accuracy_score(y_val, y_pred_val)
precision_val_gs = precision_score(y_val, y_pred_val)
recall_val_gs = recall_score(y_val, y_pred_val)
f1_val_gs = f1_score(y_val, y_pred_val)
print(f'\n')
print(f'Accuracy Val: {accuracy_val_gs}')
print(f'Precision Val: {precision_val_gs}')
print(f'Recall Val: {recall_val_gs}')
print(f'f1 Val: {f1_val_gs}')
print(f'\n')
print(f'Resultados del modelo original max_depth=100')
print(f'\n')
print(f'Accuracy Val: {accuracy_train_o}')
print(f'Precision Val: {precision_train_o}')
print(f'Recall Val: {recall_train_o}')
print(f'f1 Val: {f1_train_o}')
print(f'\n')
print(f'Accuracy Val: {accuracy_test_o}')
print(f'Precision Val: {precision_test_o}')
print(f'Recall Val: {recall_test_o}')
print(f'f1 Val: {f1_test_o}')
print(f'\n')
print(f'Accuracy Val: {accuracy_val_o}')
print(f'Precision Val: {precision_val_o}')
print(f'Recall Val: {recall_val_o}')
print(f'f1 Val: {f1_val_o}')
```

```
Resultados del modelo con los hiperparámetros del GridSearch


Accuracy Train: 0.8971061093247589
Precision Train: 0.9230769230769231
Recall Train: 0.8
f1 Train: 0.8571428571428571
```

```
Accuracy Test: 0.8432835820895522
Precision Test: 0.8636363636363636
Recall Test: 0.7169811320754716
f1 Test: 0.7835051546391751


Accuracy Val: 0.7819548872180451
Precision Val: 0.6956521739130435
Recall Val: 0.6808510638297872
f1 Val: 0.6881720430107526


Resultados del modelo original max_depth=100


Accuracy Val: 0.9855305466237942
Precision Val: 0.9914893617021276
Recall Val: 0.9708333333333333
f1 Val: 0.9810526315789473


Accuracy Val: 0.7910447761194029
Precision Val: 0.7551020408163265
Recall Val: 0.6981132075471698
f1 Val: 0.7254901960784315


Accuracy Val: 0.7218045112781954
Precision Val: 0.5925925925925926
Recall Val: 0.6808510638297872
f1 Val: 0.6336633663366336
```

Como podemos observar, el desempeño de nuestro modelo disminuyó con respecto a la evaluación del entrenamiento de nuestro modelo original (max_depth=100). Sin embargo, esto en realidad es algo bueno, ya que como nuestro modelo presentaba un claro overfitting, haciendo el uso de los parámetros dados por GridSearch obtenemos un modelo que disminuye notoriamente en el sesgo.

Estos hiperparámetros impiden que el modelo sobreaprenda los datos de entrenamiento y por ende, su ajuste al resto de los conjuntos sea mejor. Aunque nuestro modelo aún presenta algo de sesgo, es mucho menor, ya que la varianza entre el train-test y el train-val es mucho menor.

12.2 Técnica 2: RidgeClassifier

Aplicaremos la técnica de RidgeClassifier a nuestro modelo original (max_depth=100). El RidgeClassifier es un modelo de regresión logística que se aplica a nuestros datos. Aplicando esto y con la función voting_classifier se hace una combinación de ambos modelos (decision tree y ridege classifier) con el objetivo de tomar la clasificación más frecuente que se realice entre ambos.

[29]:
```
from sklearn.linear_model import RidgeClassifier
from sklearn.ensemble import VotingClassifier

# Crea modelos individuales
decision_tree = DecisionTreeClassifier(max_depth=100, random_state=42)
ridge_classifier = RidgeClassifier(alpha=1.0, random_state=42)

# Crea un clasificador de votación
voting_classifier = VotingClassifier(estimators=[('dt', decision_tree), ('rc',␣
 ↪ridge_classifier)], voting='hard')

# Entrena el clasificador de votación
voting_classifier.fit(X_train, y_train)
```

[29]:
```
VotingClassifier(estimators=[('dt',
                              DecisionTreeClassifier(max_depth=100,
                                                     random_state=42)),
                             ('rc', RidgeClassifier(random_state=42))])
```

[30]:
```
# Hacer predicciones en el conjunto de entrenamiento
y_pred_train = voting_classifier.predict(X_train)

# Hacer predicciones en el conjunto de prueba
y_pred_test = voting_classifier.predict(X_test)

# Hacer predicciones en el conjunto de validación
y_pred_val = voting_classifier.predict(X_val)

print(f'Resultados del modelo con RidgeClassifier')
# Evaluar el rendimiento del modelo (train)
accuracy = accuracy_score(y_train, y_pred_train)
precision = precision_score(y_train, y_pred_train)
recall = recall_score(y_train, y_pred_train)
f1 = f1_score(y_train, y_pred_train)
print(f'\n')
print(f'Accuracy Train: {accuracy}')
print(f'Precision Train: {precision}')
print(f'Recall Train: {recall}')
print(f'f1 Train: {f1}')

# Evaluar el rendimiento del modelo (test)
accuracy = accuracy_score(y_test, y_pred_test)
precision = precision_score(y_test, y_pred_test)
recall = recall_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test)
print(f'\n')
print(f'Accuracy Test: {accuracy}')
```

```
print(f'Precision Test: {precision}')
print(f'Recall Test: {recall}')
print(f'f1 Test: {f1}')

# Evaluar el rendimiento del modelo (val)
accuracy = accuracy_score(y_val, y_pred_val)
precision = precision_score(y_val, y_pred_val)
recall = recall_score(y_val, y_pred_val)
f1 = f1_score(y_val, y_pred_val)
print(f'\n')
print(f'Accuracy Val: {accuracy}')
print(f'Precision Val: {precision}')
print(f'Recall Val: {recall}')
print(f'f1 Val: {f1}')
print(f'\n')
print(f'Resultados del modelo original max_depth=100')
print(f'\n')
print(f'Accuracy Val: {accuracy_train_o}')
print(f'Precision Val: {precision_train_o}')
print(f'Recall Val: {recall_train_o}')
print(f'f1 Val: {f1_train_o}')
print(f'\n')
print(f'Accuracy Val: {accuracy_test_o}')
print(f'Precision Val: {precision_test_o}')
print(f'Recall Val: {recall_test_o}')
print(f'f1 Val: {f1_test_o}')
print(f'\n')
print(f'Accuracy Val: {accuracy_val_o}')
print(f'Precision Val: {precision_val_o}')
print(f'Recall Val: {recall_val_o}')
print(f'f1 Val: {f1_val_o}')
```

```
Resultados del modelo con RidgeClassifier


Accuracy Train: 0.882636655948553
Precision Train: 0.9883040935672515
Recall Train: 0.7041666666666667
f1 Train: 0.8223844282238444


Accuracy Test: 0.835820895522388
Precision Test: 0.8974358974358975
Recall Test: 0.660377358490566
f1 Test: 0.7608695652173912
```

```
Accuracy Val: 0.7819548872180451
Precision Val: 0.7368421052631579
Recall Val: 0.5957446808510638
f1 Val: 0.6588235294117647


Resultados del modelo original max_depth=100


Accuracy Val: 0.9855305466237942
Precision Val: 0.9914893617021276
Recall Val: 0.9708333333333333
f1 Val: 0.9810526315789473


Accuracy Val: 0.7910447761194029
Precision Val: 0.7551020408163265
Recall Val: 0.6981132075471698
f1 Val: 0.7254901960784315


Accuracy Val: 0.7218045112781954
Precision Val: 0.5925925925925926
Recall Val: 0.6808510638297872
f1 Val: 0.6336633663366336
```

Como podemos observar, aplicando el método del RidgeClassifier logramos que el modelo disminuya en su desempeño favoreciendo a disminuir el sesgo y varianza que observamos en el modelo original del entrenamiento. Nuevamente, las métricas obtenidas para cada conjunto de datos se encuentran más cercanas entre sí y son más realistas. Aunque nuetsro modelo sigue sin tener el desempeño esperado.

12.3 Técnica 3: Cross Validation

Como tercera técnica se utilizará el cross validation que nos servirá para evaluar el rendimiento del modelo en múltiples divisiones de los datos. En este caso, se partirán los datos 5 veces (cv=5) implicando que el modelo se entrenará en 5 ocasiones y se evaluará 5 veces para evitar que sobreaprenda un solo conjunto de datos.

Primero utilizaremos el crossvalidation para evaluar nuestro modelo original (max_depth=100).

Luego lo usaremos para evaluar nuestro modelo con los hiperparámetros dados por GridSearch para realizar una comparativa.

[32]:
```
from sklearn.model_selection import cross_val_score

#Modelo Original
# Crea el modelo de DecisionTreeClassifier con tus hiperparámetros
classifier = DecisionTreeClassifier(max_depth=100, random_state=42)
```

```
# Aplica validación cruzada con 5 divisiones (o el número deseado de divisiones)
scores = cross_val_score(classifier, X, y, cv=5, scoring='accuracy')

print("Cross Validation - Modelo Original")
# Imprime los puntajes de precisión en cada división
print("Puntajes de precisión en cada división:", scores)

# Calcula y muestra la precisión promedio
print("Precisión promedio:", scores.mean())
```

```
Cross Validation - Modelo Original
Puntajes de precisión en cada división: [0.73595506 0.75842697 0.79775281
0.75842697 0.79661017]
Precisión promedio: 0.7694343934488669
```

[33]:
```
#Modelo GridSearch
# Crea el modelo de DecisionTreeClassifier con tus hiperparámetros
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=10,␣
 ↪random_state=42, min_samples_leaf=2, min_samples_split=10)

# Aplica validación cruzada con 5 divisiones (o el número deseado de divisiones)
scores = cross_val_score(classifier, X, y, cv=5, scoring='accuracy')

print("Cross Validation - Modelo GridSearch")
# Imprime los puntajes de precisión en cada división
print("Puntajes de precisión en cada división:", scores)

# Calcula y muestra la precisión promedio
print("Precisión promedio:", scores.mean())
```

```
Cross Validation - Modelo GridSearch
Puntajes de precisión en cada división: [0.7752809  0.79775281 0.80898876
0.81460674 0.85310734]
Precisión promedio: 0.8099473116231828
```

Como podemos ver, el modelo realizado con los hiperparámetros obtenidos con el GridSearch claramente es mejor que nuestro modelo original. Ya que este no presenta un nivel grave de overfitting y la varianza es mucho menor entre las métricas de nuestros tres conjuntos de datos.