Momento de Retroalimentación: Módulo 2 Análisis y Reporte sobre el desempeño del modelo. (Portafolio Análisis) Alumno: Raul Alejandro Olivares A01752057
Modelo Seleccionado: Clasification Tree (Sklearn)

1.Dataset

Los arboles de decisión permiten una apmlia gama de datasets; ya que poseen la caracteristica de resolver problemas de clasificación y de regresión. Tambien tienen la facilidad de ajustarse a las caracteristicas del dataset, lo que convierte a este modelo una opción a considerar en diversos escenarios. El dataset con el que trabajare será el de wine el cual se encuentra en sklearn, este dataset es ideal para un arbol de clasificación puesto que el problema yace en asignarle una clase a una serie de vinos con ciertas caracteristicas; a pesar de que el dataset sea para principiantes al usarlo puede ser tan complejo como lo requiera el usuario, en otras palabras, permite que el usuario pruebe tecnicas de preprocesamiento y ajustes en los hiper parametros, ya que los datos responden a los ajustes, dando retroalimentación al usuario. Por ultimo, este dataset posee los valores reales con los cuales se podran comparar las predicciones del modelo, siendo esto una herramienta util de visualizacion para el usuario, ya que permite observar que las clases se asignan correctamente o incorrectamente segun sea el caso.

In [ ]:
```
from sklearn.datasets import load_wine
wine = load_wine()
X = wine.data
y = wine.target
```

2.Separación del Dataset

Lo siguiente a realizar es la separación del dataset en 3 conjuntos Training/Test/Validation, se suelen separar los datos en entrenamiento y pruebas con una proporción de 80/20, esto debido a varias razones como por ejemplo; equilibrio de los datos, reducción de riesgo de overfitting, eficiencia de calculo, K-fold, siendo algunas de las razones. A continuación se describen las caracteristicas y funcionalidades de cada conjunto:

° Conjunto de entrenamiento: Usado para entrenar el modelo.

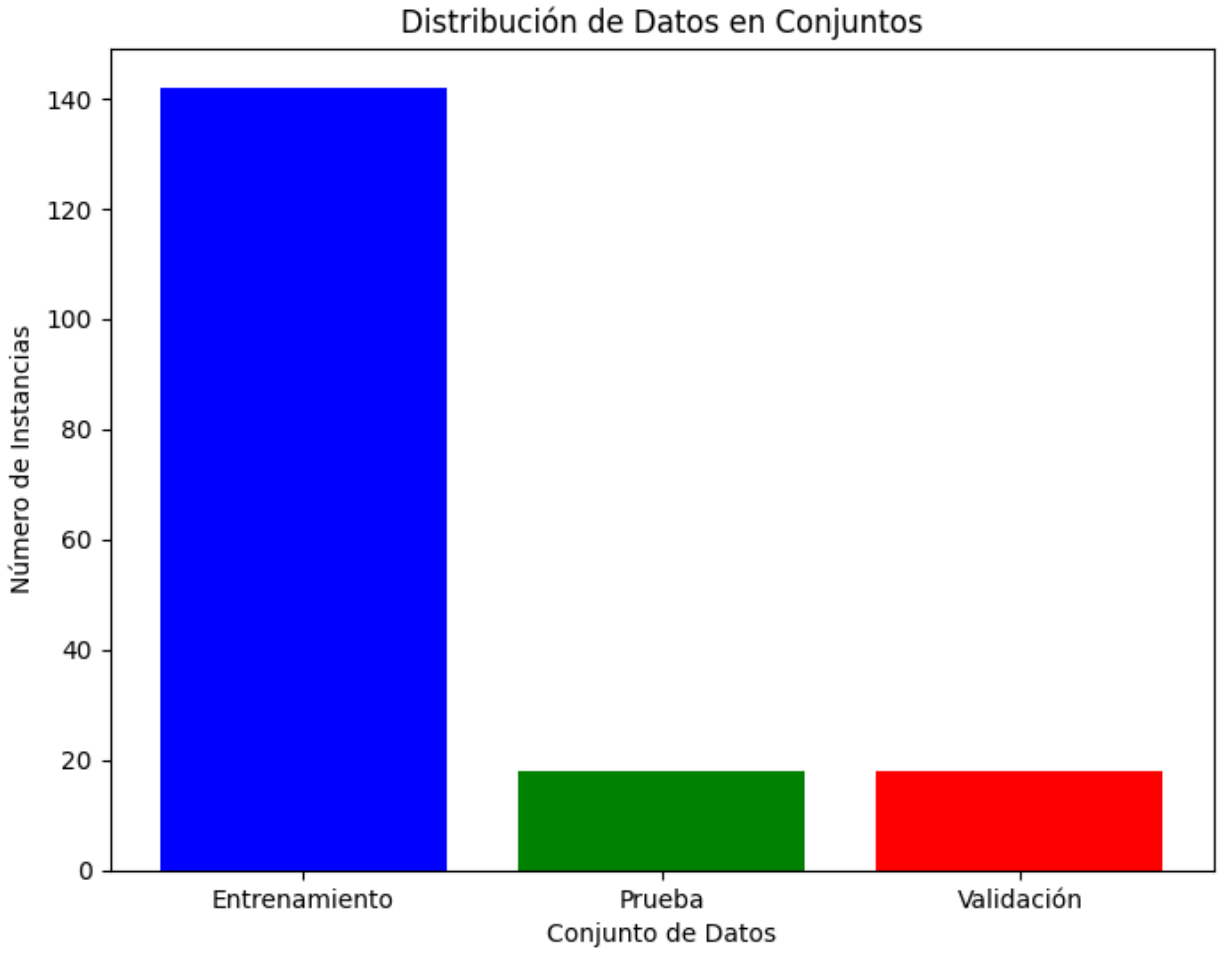
° Conjunto de prueba: Usado para evaluar el rendimiento del modelo después del entrenamiento.

° Conjunto de validación: Usardo para ajustar hiperparámetros y realizar validación adicional.

2.1 Gráfico de distribución de los sub datasets

In [ ]:

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Divide los datos en un conjunto de entrenamiento (80%) y un conjunto temporal (20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=

# Divide el conjunto temporal en un conjunto de prueba (50%) y un conjunto de validaci
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_

#Grafico de barras para mostrar la proporción de los conjuntos de datos
#Etiquetas para las barras
labels = ['Entrenamiento', 'Prueba', 'Validación']

# Número de instancias en cada conjunto
sizes = [len(X_train), len(X_test), len(X_val)]

# Colores para los conjuntos
colors = ['blue', 'green', 'red']

# Crea un gráfico de barras
plt.figure(figsize=(8, 6))
plt.bar(labels, sizes, color=colors)
plt.xlabel('Conjunto de Datos')
plt.ylabel('Número de Instancias')
plt.title('Distribución de Datos en Conjuntos')
plt.show()
```



2.2 Sub dataset de entrenamiento

In [ ]:
```
#Datos dentro del conjunto de entrenamiento
import pandas as pd

target = pd.Series(y_train, name='target')
df = pd.DataFrame(X_train, columns=wine.feature_names)
df = pd.concat([df, target], axis=1)
print(f"Total de datos: {len(X)}\n Datos dentro del conjunto de entrenamiento: {len(df
df.head(10)
```

```
Total de datos: 178
 Datos dentro del conjunto de entrenamiento: 142
```

Out[ ]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phe
0	14.34	1.68	2.70	25.0	98.0	2.80	1.31	
1	12.53	5.51	2.64	25.0	96.0	1.79	0.60	
2	12.37	1.07	2.10	18.5	88.0	3.52	3.75	
3	13.48	1.67	2.64	22.5	89.0	2.60	1.10	
4	13.07	1.50	2.10	15.5	98.0	2.40	2.64	
5	12.22	1.29	1.94	19.0	92.0	2.36	2.04	
6	12.67	0.98	2.24	18.0	99.0	2.20	1.94	
7	13.34	0.94	2.36	17.0	110.0	2.53	1.30	
8	13.62	4.95	2.35	20.0	92.0	2.00	0.80	
9	13.16	2.36	2.67	18.6	101.0	2.80	3.24	

2.2 Sub dataset de pruebas

In [ ]:
```
#Datos dentro del conjunto de pruebas

target = pd.Series(y_test, name='target')
df = pd.DataFrame(X_test, columns=wine.feature_names)
df = pd.concat([df, target], axis=1)
print(f"Total de datos: {len(X)}\n Datos dentro del conjunto de pruebas: {len(df.index
df.head(10)
```

```
Total de datos: 178
 Datos dentro del conjunto de pruebas: 18
```

Out[ ]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phe
0	13.50	3.12	2.62	24.0	123.0	1.40	1.57	
1	14.21	4.04	2.44	18.9	111.0	2.85	2.65	
2	12.77	2.39	2.28	19.5	86.0	1.39	0.51	
3	12.93	2.81	2.70	21.0	96.0	1.54	0.50	
4	13.58	1.66	2.36	19.1	106.0	2.86	3.19	
5	13.73	1.50	2.70	22.5	101.0	3.00	3.25	
6	13.16	3.57	2.15	21.0	102.0	1.50	0.55	
7	13.40	4.60	2.86	25.0	112.0	1.98	0.96	
8	12.37	1.21	2.56	18.1	98.0	2.42	2.65	
9	13.50	1.81	2.61	20.0	96.0	2.53	2.61	

2.2 Sub dataset de validacion

In [ ]:

```
#Datos del conjunto de validacion

target = pd.Series(y_val, name='target')
df = pd.DataFrame(X_val, columns=wine.feature_names)
df = pd.concat([df, target], axis=1)
print(f"Total de datos: {len(X)}\n Datos dentro del conjunto de validación: {len(df.ir
df.head(10)
```

```
Total de datos: 178
 Datos dentro del conjunto de validación: 18
```

Out[ ]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phe
0	13.88	1.89	2.59	15.0	101.0	3.25	3.56	
1	12.77	3.43	1.98	16.0	80.0	1.63	1.25	
2	12.42	1.61	2.19	22.5	108.0	2.00	2.09	
3	14.02	1.68	2.21	16.0	96.0	2.65	2.33	
4	11.41	0.74	2.50	21.0	88.0	2.48	2.01	
5	12.37	1.63	2.30	24.5	88.0	2.22	2.45	
6	13.11	1.01	1.70	15.0	78.0	2.98	3.18	
7	13.41	3.84	2.12	18.8	90.0	2.45	2.68	
8	11.61	1.35	2.70	20.0	94.0	2.74	2.92	
9	12.08	1.13	2.51	24.0	78.0	2.00	1.58	

3.Calculo de Bias o Sesgo

El bias se define como la diferencia entre las predicciones del modelo y los valores reales en un conjunto de datos independiente, que el modelo no ha visto durante el entrenamiento.
El bias se clásifica en tres categorías bajo, medio o alto; estas clasificaciones son utilizadas para analizar el grado de generalización que presenta el modelo, las caracteristicas de cada clasificación son las siguientes:
° Bajo: El modelo posee una alta capacidad de ajuste a datos de entrenamiento, generalmente no produce underfitting, realiza predicciones precisas para los datos de entrenamiento.
° Medio: El modelo es capaz de entender patrones, puede generalizar de buena forma en un conjunto de datos de prueba.
° Alto: El modelo podría omitir patrones importantes, el modelo tiende a un underfitting, tiende a tener un rendimiento ineficiente en los datos de entrenamiento y prueba.

In [ ]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import numpy as np

# Creación del arbol
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Predicciones de acuerdo a cada conjunto
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
y_val_pred = model.predict(X_val)

# Precisiónes de acuerdo a cada prediccion
accuracy_train = accuracy_score(y_train, y_train_pred)
accuracy_test = accuracy_score(y_test, y_test_pred)
accuracy_val = accuracy_score(y_val, y_val_pred)

# Calculo del sesgo y la varianza
bias = accuracy_train - accuracy_test  # Sesgo: Identifica si se generaliza de forma c

# Clasificación del nivel de sesgo
if bias < 0.1:
    bias_classification = "Bajo"
elif bias < 0.3:
    bias_classification = "Medio"
else:
    bias_classification = "Alto"
```

```
Precisión en entrenamiento: 1.0
Precisión en prueba: 0.8888888888888888
Precisión en validación: 1.0
Sesgo: 0.11111111111111116
Clasificación del sesgo: Medio
```

4.Calculo de Varianza

La varianza se refiere a la variabilidad de las predicciones del modelo cuando se le presenta diferentes conjuntos de datos de prueba. Cuanto mayor sea la varianza, más cambian las predicciones del modelo en respuesta a diferentes conjuntos de datos de prueba y se calcula

obteniendo el rendimiento del modelo en diferentes conjuntos de datos. Al igual que el sesgo se clásifica en 3 tipos:
° Bajo: Un modelo con varianza baja no se ve afectado por la entrada de nuevos datos, realiza suposiciones simplificadas por lo que no podría detectar patrones complejos y suele tender al underfitting.
° Medio: Un modelo con varianza media logra un ajuste entre el ajuste de los datos de entrenamiento y su generalización, lo que le permite ofrecer un rendimiento consistente.
° Alto: Un modelo con un grado alto de varianza se convierte en un modelo sensible a la entrada de datos; se tiende a un sobre ajuste, lo que genera un buen rendimiento en datos de entrenamiento y una deficiencia en datos de prueba.

In [ ]:
```
variance = np.var([accuracy_train, accuracy_test, accuracy_val])  # Varianza
# Clasificación del nivel de varianza
if variance < 0.1:
    variance_classification = "Bajo"
elif variance < 0.3:
    variance_classification = "Medio"
else:
    variance_classification = "Alto"
```

5.Nivel de Ajuste del Modelo

El nivel de ajuste del modelo es una medida que evalúa cómo se comporta un modelo de aprendizaje automático en términos de sesgo y varianza. Es util para encontrar un equilibrio entre las dos metricas anteriores. Se clasifica en tres tipos; Overfitting, fitting y Underfitting. A continuación se presenta las implicaciónes de cada clase:
° Overfitting: El modelo se ajusta demasiado a los datos de entrenamiento, lo que provoca que tenga deficiencias en los datos de prueba. (Baja varianza y alto Sesgo)
° Fitting: El modelo generaliza de forma correcta, se encuentra un balance entre el sesgo y la varianza, se tiene un buen rendimiento en los datos de entrenamiento y prueba.
° Underfitting: El modelo generaliza de forma simple, lo que provoca que no encuentre patrones complejos y que tenga rendimientos deficientes (Alto sesgo y baja varianza).

In [ ]:
```
# Nivel de ajuste del modelo
if bias_classification == "Bajo" and variance_classification == "Bajo":
    model_fit = "Adecuado (Ajuste adecuado)"
elif bias_classification == "Alto" or variance_classification == "Alto":
    model_fit = "Subajuste (Underfitting)"
else:
    model_fit = "Sobreajuste (Overfitting)"
```

6.Analisis sobre el desempeño

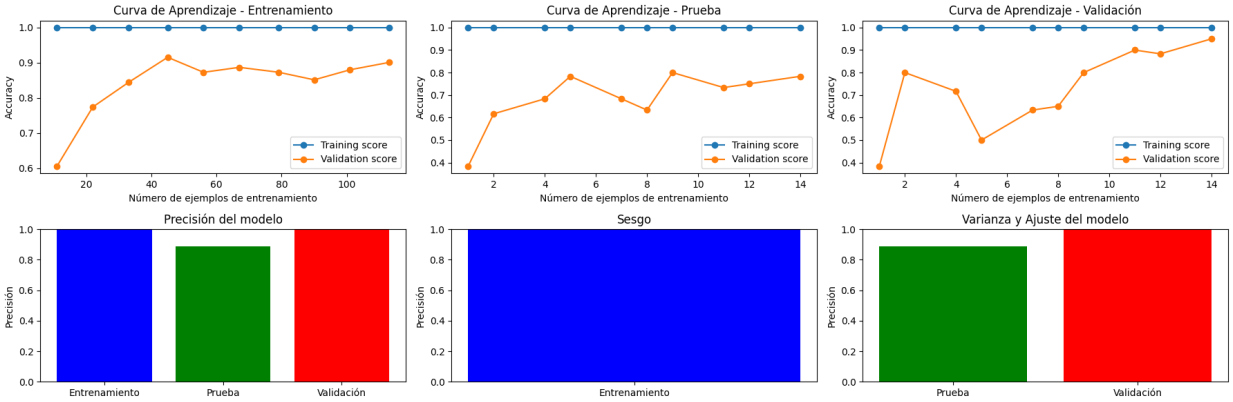
A continuación se muestran los valores de precisión, sesgo y varianza obtenidos dentro de las predicciones; ademas de ello tambien se incluyen una serie de gráficos para explicar como se visualiza el sesgo la varianza y el nivel de ajuste de nuestro modelo en nuestros datos.

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import learning_curve

# Crea una función para trazar curvas de aprendizaje
def plot_learning_curve(X, y, model, title):
    train_sizes, train_scores, test_scores = learning_curve(model, X, y, cv=5, scoring

    plt.plot(train_sizes, np.mean(train_scores, axis=1), 'o-', label='Training score')
    plt.plot(train_sizes, np.mean(test_scores, axis=1), 'o-', label='Validation score'

    plt.xlabel('Número de ejemplos de entrenamiento')
    plt.ylabel('Accuracy')
    plt.legend(loc='best')
    plt.title(title)

# Colores para las barras
colores = ['blue', 'green', 'red']

# Graficar curvas de aprendizaje y métricas de sesgo, varianza y ajuste en una ventana
plt.figure(figsize=(18, 6))

# Curvas de aprendizaje
plt.subplot(231)
plot_learning_curve(X_train, y_train, model, 'Curva de Aprendizaje - Entrenamiento')
plt.subplot(232)
plot_learning_curve(X_test, y_test, model, 'Curva de Aprendizaje - Prueba')
plt.subplot(233)
plot_learning_curve(X_val, y_val, model, 'Curva de Aprendizaje - Validación')

# Visualizar sesgo, varianza y nivel de ajuste
plt.subplot(234)
plt.bar(['Entrenamiento', 'Prueba', 'Validación'], [accuracy_train, accuracy_test, acc
plt.ylim(0, 1)
plt.ylabel('Precisión')
plt.title('Precisión del modelo')

# Métricas de sesgo, varianza y ajuste
plt.subplot(235)
plt.bar(['Entrenamiento'], [accuracy_train], color=colores[0])
plt.ylim(0, 1)
plt.ylabel('Precisión')
plt.title('Sesgo')

plt.subplot(236)
plt.bar(['Prueba', 'Validación'], [accuracy_test, accuracy_val], color=colores[1:])
plt.ylim(0, 1)
plt.ylabel('Precisión')
plt.title('Varianza y Ajuste del modelo')

plt.tight_layout()
plt.show()

# Imprime los resultados
print("-"*30)
print("Reporte sobre del desempeño: ")
print("\tPrecisión en entrenamiento:", accuracy_train)
print("\tPrecisión en prueba:", accuracy_test)
```

```
print("\tPrecisión en validación:", accuracy_val)
print("\tSesgo:", bias)
print("\tClasificación del sesgo:", bias_classification)
print("\tVarianza:", variance)
print("\tClasificación de la varianza:", variance_classification)
print("\tNivel de ajuste del modelo:", model_fit)
```

```
c:\Users\rauli\anaconda3\envs\IA\lib\site-packages\sklearn\model_selection\_split.py:
700: UserWarning: The least populated class in y has only 4 members, which is less th
an n_splits=5.
  warnings.warn(
c:\Users\rauli\anaconda3\envs\IA\lib\site-packages\sklearn\model_selection\_split.py:
700: UserWarning: The least populated class in y has only 1 members, which is less th
an n_splits=5.
  warnings.warn(
```




```
------------------------------
Reporte sobre del desempeño:
        Precisión en entrenamiento: 1.0
        Precisión en prueba: 0.8888888888888888
        Precisión en validación: 1.0
        Sesgo: 0.11111111111111116
        Clasificación del sesgo: Medio
        Varianza: 0.002743484224965709
        Clasificación de la varianza: Bajo
        Nivel de ajuste del modelo: Sobreajuste (Overfitting)
```

6.1 Análisis

6.1.1 Sesgo

El sesgo del modelo presenta un valor de un sesgo medio; recordemos que el sesgo se calcula a partir de las predicciones elaboradas con los datos de entrenamiento y con los datos de prueba, por lo tanto, lo que genera que la clasificacion del sesgo sea media es el rendimiento de nuestro modelo con los datos de prueba; como se puede observar en la gráfica de precisión del modelo, a pesar de que nuestro modelo esta realizando una predicción "aceptable" para los datos de pruebas el valor de precisión que el modelo posee en los datos de entrenamiento es demsaidado elevado. Apoyando este punto, se observa lo mismo com las curvas de aprendizaje ya que el modelo se ajusta perfectamente a los datos de entrenamiento pero al ser sometido a los datos de prueba comienzan a existir anormalidades, dandonos indicios de overfitting.

6.1.2 Varianza

La varianza de nuestro modelo es baja, lo que significa que las clasificaciones estan resultando ser similares para nuestros tres sub datasets, apoyandome en los gráficos podemos notar este comportamiento en el varianza y ajuste del modelo, en el cual se observa que los puntajes de precisión son bastante similares, lo que confirma que nuestro dataset esta clasificando de forma correcta con distintos datos.

6.1.3 Nivel de ajuste del modelo

Por ultimo tenemos el ajuste del modelo; en este caso se presenta un ajuste de overfitting y podemos comprobarlo facilmente con las curvas de aprendizaje, el modelo se adapta demasiado bien a los datos de entrenamiento pero con los datos de prueba hay rendimientos deficientes. A continuación veremos 3 técnicas utiles para mejorar el desempeño.

7.Técnicas para mejorar el desempeño

7.1 Modificacion de Hiperparametros

Para la elaboración de la siguiente tecnica se aleatorizaron los hiperparametros que pueden tener un efecto positivo en redurcir el overfitting, se establece un minimo valor para la precisión deseada y el programa se encarga de buscar algun arbol que cumpla con estas caracteristicas. En este caso los datos respondieron de forma positiva a la tecnica logrando cambiar el overfitting por fitting.

In [ ]:
```
#Tecnica 1. Modificacion de Hiperparametros
import random
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import random

# Umbral de precisión definido por el usuario (por ejemplo, 0.90 para el 90%)
umbral_precisio = 0.94

# Hiperparámetros iniciales
max_depth = random.randint(1, 20)
min_samples_split = random.randint(2, 10)
min_samples_leaf = random.randint(1, 10)

mejor_modelo = None
mejor_accuracy_entrenamiento = 0
mejor_accuracy_prueba = 0

#Maximo de iteraciones para buscar un modelo que cumpla con el umbral
max_iteration=1000

while True:
    # Crear y entrenar el modelo con los hiperparámetros actuales
```

```
    modelo = DecisionTreeClassifier(
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        random_state=42  # Ajusta una semilla aleatoria para resultados reproducibles
    )
    modelo.fit(X_train, y_train)

    # Realiza predicciones en datos de entrenamiento y prueba
    y_pred_train = modelo.predict(X_train)
    y_pred_test = modelo.predict(X_test)

    # Calcula la precisión en datos de entrenamiento y prueba
    accuracy_entrenamiento = accuracy_score(y_train, y_pred_train)
    accuracy_prueba = accuracy_score(y_test, y_pred_test)

    # Verifica si se alcanzó el umbral de precisión
    if accuracy_entrenamiento >= umbral_precisio and accuracy_prueba >= umbral_precisi
        mejor_modelo = modelo
        mejor_accuracy_entrenamiento = accuracy_entrenamiento
        mejor_accuracy_prueba = accuracy_prueba
        break
    max_iteration =max_iteration-1
    if(max_iteration <=0):
        print('Adjust the umbral, there is no model found')
        break
    # Ajusta hiperparámetros aleatoriamente
    max_depth = random.randint(1, 20)
    min_samples_split = random.randint(2, 10)
    min_samples_leaf = random.randint(1, 10)

# El bucle terminará cuando se alcance el umbral de precisión
print("Mejor modelo alcanzó el umbral de precisión:")
print("Precisión en entrenamiento:", mejor_accuracy_entrenamiento)
print("Precisión en prueba:", mejor_accuracy_prueba)
```

```
Mejor modelo alcanzó el umbral de precisión:
Precisión en entrenamiento: 0.971830985915493
Precisión en prueba: 0.9444444444444444
```

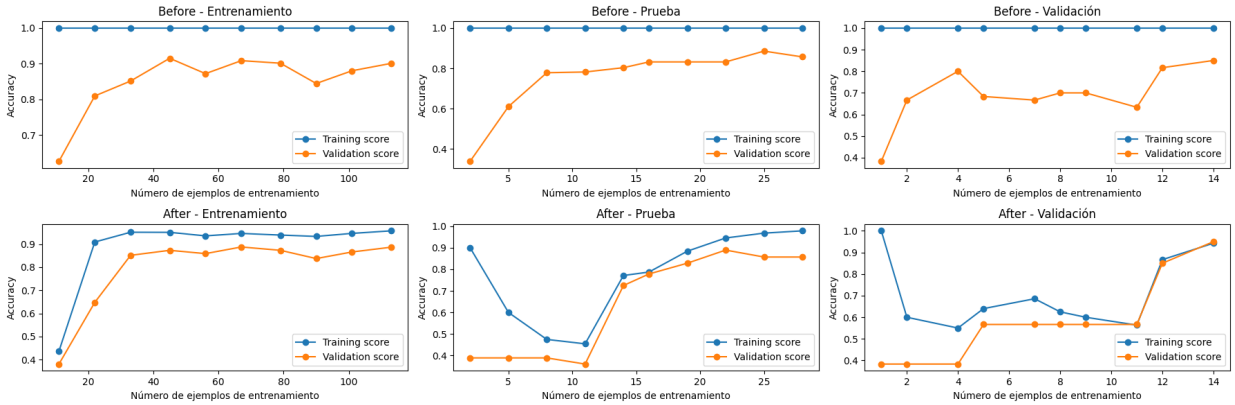
7.1.1 Construcción de gráficos comparativos e impresión del nuevo reporte de análisis

In [ ]:

```
#Tecnica 1. Modificacion de Hiperparametros

model = DecisionTreeClassifier()

plt.figure(figsize=(18, 6))

# Curvas de aprendizaje antes del ajuste de hiper parametros
plt.subplot(231)
plot_learning_curve(X_train, y_train, model, 'Before - Entrenamiento')
plt.subplot(232)
plot_learning_curve(X_test, y_test, model, 'Before - Prueba')
plt.subplot(233)
plot_learning_curve(X_val, y_val, model, 'Before - Validación')

#Curvas de aprendizaje despues del ajuste de hiper parametros
plt.subplot(234)
```

```
plot_learning_curve(X_train, y_train, mejor_modelo, 'After - Entrenamiento')
plt.subplot(235)
plot_learning_curve(X_test, y_test, mejor_modelo, 'After - Prueba')
plt.subplot(236)
plot_learning_curve(X_val, y_val, mejor_modelo, 'After - Validación')

plt.tight_layout()
plt.show()

#REPORTE DEL AJUSTE
# Predicciones de acuerdo a cada conjunto
y_train_pred = mejor_modelo.predict(X_train)
y_test_pred = mejor_modelo.predict(X_test)
y_val_pred = mejor_modelo.predict(X_val)

# Precisiónes de acuerdo a cada prediccion
accuracy_train = accuracy_score(y_train, y_train_pred)
accuracy_test = accuracy_score(y_test, y_test_pred)
accuracy_val = accuracy_score(y_val, y_val_pred)

# Calculo del sesgo y la varianza
bias = accuracy_train - accuracy_test  # Sesgo: Identifica si se generaliza de forma c

# Clasificación del nivel de sesgo
if bias < 0.1:
    bias_classification = "Bajo"
elif bias < 0.3:
    bias_classification = "Medio"
else:
    bias_classification = "Alto"

variance = np.var([accuracy_train, accuracy_test, accuracy_val])  # Varianza
# Clasificación del nivel de varianza
if variance < 0.1:
    variance_classification = "Bajo"
elif variance < 0.3:
    variance_classification = "Medio"
else:
    variance_classification = "Alto"

# Nivel de ajuste del modelo
if bias_classification == "Bajo" and variance_classification == "Bajo":
    model_fit = "Adecuado (Ajuste adecuado)"
elif bias_classification == "Alto" or variance_classification == "Alto":
    model_fit = "Subajuste (Underfitting)"
else:
    model_fit = "Sobreajuste (Overfitting)"

# Imprime los resultados
print("-"*30)
print("Reporte sobre del desempeño (Tecnica ajuste hiperparametros): ")
print("\tPrecisión en entrenamiento:", accuracy_train)
print("\tPrecisión en prueba:", accuracy_test)
print("\tPrecisión en validación:", accuracy_val)
print("\tSesgo:", bias)
print("\tClasificación del sesgo:", bias_classification)
print("\tVarianza:", variance)
print("\tClasificación de la varianza:", variance_classification)
print("\tNivel de ajuste del modelo:", model_fit)
```

```
c:\Users\rauli\anaconda3\envs\IA\lib\site-packages\sklearn\model_selection\_split.py:
700: UserWarning: The least populated class in y has only 1 members, which is less th
an n_splits=5.
  warnings.warn(
c:\Users\rauli\anaconda3\envs\IA\lib\site-packages\sklearn\model_selection\_split.py:
700: UserWarning: The least populated class in y has only 1 members, which is less th
an n_splits=5.
  warnings.warn(
```




```
--------------------------------
Reporte sobre del desempeño (Tecnica ajuste hiperparametros):
        Precisión en entrenamiento: 0.971830985915493
        Precisión en prueba: 0.9444444444444444
        Precisión en validación: 1.0
        Sesgo: 0.027386541471048576
        Clasificación del sesgo: Bajo
        Varianza: 0.0005144373068138934
        Clasificación de la varianza: Bajo
        Nivel de ajuste del modelo: Adecuado (Ajuste adecuado)
```

7.2 Poda del arbol (Regularización)

In [ ]:

```python
#Tecnica 2 poda de arbo (regularización)
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split


# Crear un árbol de decisión sin restricciones
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

# Evaluar el árbol sin podar en el conjunto de prueba
accuracy_before_pruning = tree.score(X_test, y_test)
print("Precisión antes de la poda:", accuracy_before_pruning)
model=tree

# Predicciones de acuerdo a cada conjunto
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
y_val_pred = model.predict(X_val)

# Precisiónes de acuerdo a cada prediccion
accuracy_train = accuracy_score(y_train, y_train_pred)
accuracy_test = accuracy_score(y_test, y_test_pred)
accuracy_val = accuracy_score(y_val, y_val_pred)
```

```
print(accuracy_train)
print(accuracy_test)

# Aplicar la poda basada en cost-complexity
path = tree.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

# Entrenar árboles con diferentes alphas
trees = []
for ccp_alpha in ccp_alphas:
    tree = DecisionTreeClassifier(random_state=58, ccp_alpha=ccp_alpha)
    tree.fit(X_train, y_train)
    trees.append(tree)

# Evaluar la precisión de los árboles podados en el conjunto de prueba
accuracies = [tree.score(X_test, y_test) for tree in trees]

# Encontrar el alpha óptimo
best_alpha = ccp_alphas[np.argmax(accuracies)]
print("Mejor alpha:", best_alpha)

# Crear un árbol podado con el mejor alpha
pruned_tree = DecisionTreeClassifier(random_state=42, ccp_alpha=best_alpha)
pruned_tree.fit(X_train, y_train)
y_pred_train = pruned_tree.predict(X_train)
y_pred_test = pruned_tree.predict(X_test)

# Calcular la precisión en datos de entrenamiento y prueba
accuracy_entrenamiento_aft = accuracy_score(y_train, y_pred_train)
accuracy_prueba_aft = accuracy_score(y_test, y_pred_test)

# Evaluar la precisión del árbol podado en el conjunto de prueba
accuracy_after_pruning = pruned_tree.score(X_test, y_test)
print("Precisión después de la poda:", accuracy_after_pruning)
print("1:",accuracy_entrenamiento_aft)
print("2:",accuracy_prueba_aft)
```

```
Precisión antes de la poda: 0.9444444444444444
1.0
0.9444444444444444
Mejor alpha: 0.0
Precisión después de la poda: 0.9444444444444444
1: 1.0
2: 0.9444444444444444
```