Machine Learning II: Introducción a los Métodos de Clasificación Supervisada

CONCEPTOS

PARTE 1: Un enfoque práctico para Machine Learning

1. Acerca del software.
2. ¿Qué es Machine Learning?
3. Modelando el problema de Machine Learning
4. El problema de clasificación supervisada. Un ejemplo programático básico guiado

    4.1 Representación del problema en sklearn

    4.2 Aprendiendo y prediciendo

    4.3 Más sobre el 'feature space'

    4.4 Entrenamiento y prueba

    4.5 Selección de modelos (I)

PARTE 2: Conceptos de aprendizaje y teoría

1. ¿Qué es aprender?

    5.1 Aprendizaje PAC

2. Dentro del modelo de aprendizaje

    6.1 El algoritmo de aprendizaje Machine Learning humano

    6.2 Clase de modelo y espacio de hipótesis

    6.3 Función objetivo

    6.4 Algoritmo de búsqueda/optimización/aprendizaje

3. Curvas de aprendizaje y sobreajuste (Overfitting)

    7.1 Curvas de aprendizaje

    7.2 Sobreajuste

4. Curas para el sobreajuste

    8.1 Selección de modelos (II)

8.2 Regularización

8.3 Conjunto

5. ¿Qué hacer cuando...?

PARTE 3: Primeros modelos

1. Modelos generativos y discriminativos

   10.1 Modelos bayesianos (Naive Bayes) y algunas aplicaciones.

   10.2 Máquinas de Vectores de Soporte (Support Vector Machines).

PARTE 2: Conceptos de aprendizaje y teoría

5. ¿Qué es aprender?

En machine learning, aprender significa que un modelo tiene la capacidad de identificar patrones útiles dentro de los datos disponibles y usarlos para realizar predicciones o tomar decisiones en nuevos contextos. Este proceso no es trivial y requiere de una comprensión profunda de varios conceptos clave, técnicas y herramientas que se combinan para lograr un modelo confiable y generalizable.

El aprendizaje no se limita a la mera memorización de datos. Más bien, busca lograr un equilibrio entre la capacidad de ajustarse adecuadamente a los datos observados y la capacidad de generalizar a datos nuevos. Este proceso se ve afectado por factores como la cantidad y calidad de los datos, la arquitectura del modelo, la complejidad del problema, y las estrategias utilizadas durante el entrenamiento.

Los Componentes del Proceso de Aprendizaje

Para entender qué significa aprender en machine learning, debemos analizar los siguientes aspectos esenciales:

1. **Datos de Entrenamiento:** Representan las muestras sobre las cuales el modelo aprende. Estos datos deben ser representativos del dominio del problema para garantizar que el modelo capte patrones relevantes.
2. **Modelo:** Una representación matemática que relaciona las características de entrada con las salidas deseadas. Ejemplos de modelos incluyen regresiones lineales, redes neuronales, y árboles de decisión.
3. **Función de Costo o Pérdida:** Una métrica que cuantifica qué tan mal están las predicciones del modelo comparadas con los valores reales. Minimizar esta función es el objetivo central del entrenamiento.
4. **Algoritmo de Aprendizaje:** El mecanismo utilizado para ajustar los parámetros del modelo, como el *gradient descent*, que busca iterativamente minimizar la función de pérdida.

5. **Evaluación:** Métodos para medir el desempeño del modelo en datos no utilizados durante el entrenamiento, como conjuntos de validación o prueba.

Dos conceptos fundamentales que surgen de este proceso son el **error de entrenamiento** y el **error de generalización**, que analizaremos en profundidad.

Error de Entrenamiento (Error dentro de la muestra), E_{in}

El **error de entrenamiento**, denotado como E_{in}, es la tasa de error o pérdida calculada sobre el conjunto de datos de entrenamiento. Este error refleja qué tan bien el modelo se ajusta a los datos observados durante el entrenamiento.

Importancia de E_{in}

- **Indicador de Aprendizaje Inicial:** Un E_{in} alto al inicio del entrenamiento generalmente significa que el modelo necesita más iteraciones para ajustar sus parámetros.
- **Señal de Sobreajuste:** Un E_{in} extremadamente bajo puede indicar que el modelo está capturando no solo los patrones relevantes, sino también el ruido en los datos de entrenamiento, lo que lleva al sobreajuste.

Cómo se Calcula E_{in}

En general, E_{in} se mide utilizando la misma función de pérdida utilizada durante el entrenamiento. Por ejemplo:

- En problemas de regresión, se podría usar el error cuadrático medio (*Mean Squared Error*).
- En problemas de clasificación, se puede emplear la entropía cruzada o la tasa de error.

Ejemplo Práctico: Cálculo de E_{in}

Supongamos que tenemos un modelo de regresión lineal entrenado con 100 muestras, y la función de pérdida es el error cuadrático medio:

$$E_{\text{in}} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2,$$

donde y_i son los valores reales y $\hat{y}_i$ son las predicciones del modelo. Un E_{in} bajo sugiere que el modelo está ajustándose bien a estos datos.

Error de Generalización (Error de Prueba), E_{out}

El **error de generalización**, denotado como E_{out}, mide la capacidad del modelo para hacer predicciones precisas en datos no vistos previamente. Este error es una representación directa de la utilidad del modelo en aplicaciones prácticas.

Importancia de E_{out}

- **Evalúa la Capacidad del Modelo para Generalizar:** Un modelo con un E_{out} bajo tiene mayor probabilidad de funcionar bien en entornos reales.

- **Detecta Sobreajuste o Subajuste:** Si E_{out} es significativamente mayor que E_{in}, el modelo podría estar sobreajustado. Si ambos errores son altos, el modelo podría estar subajustado.

Cómo se Calcula E_{out}

E_{out} generalmente no puede calcularse directamente porque no conocemos todas las posibles variaciones de datos no vistos. En su lugar, lo aproximamos utilizando:

- **Conjuntos de Validación o Prueba:** Dividimos los datos disponibles en conjuntos separados para evaluación.
- **Validación Cruzada:** Alternamos las particiones de datos para realizar múltiples evaluaciones.

Ejemplo Práctico: Cálculo de E_{out}

Imagina que tenemos un conjunto de prueba con 50 muestras, y utilizamos la misma función de pérdida que para E_{in}. Calculamos el error promedio en este conjunto para estimar E_{out}:

$$E_{\text{out}} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \left(y_i - \hat{y}_i \right)^2 .$$

La Relación Entre E_{in} y E_{out}

La relación entre estos errores es crucial para entender la efectividad del modelo. Matemáticamente, sabemos que:

$$E_{\text{out}} \geq E_{\text{in}} .$$

Intuición

- El modelo siempre tendrá un desempeño al menos tan bueno en los datos de entrenamiento como en los datos no vistos.
- Esto se debe a que los datos de entrenamiento son conocidos y el modelo está ajustado específicamente para ellos.

Desafíos y Estrategias

- **Reducir E_{out}:** La meta principal del aprendizaje es minimizar E_{out}. Esto implica crear modelos que no solo sean precisos en los datos de entrenamiento, sino también en datos nuevos.
- **Cerrar la Brecha entre E_{in} y E_{out}:** Esto se logra limitando la complejidad del modelo y utilizando técnicas como la regularización.

Estrategias para Mejorar la Generalización

1. **Aumentar la Variedad y Cantidad de Datos:** Más datos generalmente ayudan al modelo a captar patrones generales y evitar el sobreajuste.
2. **Regularización:** Penaliza la complejidad del modelo para evitar que se ajuste demasiado a los datos de entrenamiento.
3. **Validación Cruzada:** Estima el error de generalización de manera más robusta y confiable.

4. **Reducción de Dimensionalidad:** Técnicas como PCA eliminan características redundantes o irrelevantes que pueden confundir al modelo.

Validación Cruzada: Herramienta Fundamental

La validación cruzada es una técnica que divide los datos en múltiples particiones, alternando entre conjuntos de entrenamiento y validación. Esto permite obtener una estimación más precisa de E_{out}.

Variantes Comunes de Validación Cruzada

1. **K-Fold Cross Validation:** Divide los datos en k partes, entrenando y validando k veces, utilizando un subconjunto diferente en cada iteración.
2. **Leave-One-Out Cross Validation (LOOCV):** Cada muestra actúa como un conjunto de validación una vez, maximizando el uso de los datos disponibles.
3. **Stratified Cross Validation:** Divide los datos preservando la proporción de clases, ideal para problemas de clasificación desbalanceados.

Aprender en machine learning es un proceso iterativo que busca minimizar el error de generalización mientras se asegura de que el modelo no se ajuste excesivamente a los datos de entrenamiento. Comprender y gestionar la relación entre E_{in} y E_{out} es fundamental para crear modelos efectivos y confiables. Con técnicas como la validación cruzada, la regularización y el aumento de datos, podemos mejorar significativamente la capacidad de los modelos para generalizar, logrando soluciones prácticas y robustas para problemas reales.

Aprendizaje Probablemente Aproximadamente Correcto (PAC)

En el ámbito del machine learning, el **Aprendizaje Probablemente Aproximadamente Correcto (Probably Approximately Correct, PAC)** es un marco teórico que busca responder preguntas fundamentales sobre la capacidad de un modelo para generalizar correctamente a nuevos datos. Este enfoque proporciona una base matemática para analizar y cuantificar el error de generalización, E_{out}, en relación con el error de entrenamiento, E_{in}, el tamaño del conjunto de datos (N), y la complejidad del modelo (C). Al hacerlo, el aprendizaje PAC define un límite probabilístico sobre el desempeño del modelo, vinculando conceptos como la complejidad, el tamaño de muestra, y la precisión.

Conceptos Fundamentales del Aprendizaje PAC

1. **"Probablemente"**: Indica que las garantías teóricas sobre el desempeño del modelo son válidas con una alta probabilidad, típicamente denotada como $(1-\delta)$, donde δ es un pequeño valor predefinido.
2. **"Aproximadamente"**: Señala que no necesariamente alcanzaremos el error óptimo, pero el modelo estará lo suficientemente cerca de él dentro de un margen tolerable, ϵ.
3. **Correcto**: Significa que el modelo generaliza razonablemente bien en nuevos datos, cumpliendo con los límites establecidos para ϵ y δ.

Caracterización de la Brecha del Error de Generalización, Ω

Una de las contribuciones clave de la teoría PAC es la descripción precisa de la brecha entre el error de entrenamiento y el error de generalización. Esta brecha, Ω, depende de la cantidad de datos (N), la complejidad del modelo (C), y el nivel de confianza deseado $(1-\delta)$.
Matemáticamente, se puede expresar como:

$$E_{\text{out}} \leq E_{\text{in}} + O\left(\sqrt{\frac{\log C}{N}}\right),$$

donde:

- C representa la **complejidad del espacio de hipótesis**, también conocida como capacidad del modelo o riqueza de las funciones posibles.
- N es el número de muestras de entrenamiento.
- O indica que el término decrece con el tamaño de muestra y depende logarítmicamente de la complejidad.

Intuición Tras la Fórmula

- A mayor complejidad del modelo (C), la brecha Ω será mayor. Esto refleja el riesgo de sobreajuste asociado a modelos complejos.
- A medida que aumenta el número de muestras (N), Ω disminuye. Esto resalta la importancia de contar con datos suficientes para garantizar una buena generalización.

El Papel de la Complejidad del Modelo y el Tamaño de la Muestra

El aprendizaje PAC establece que el desempeño del modelo está estrechamente vinculado con dos factores fundamentales: **la complejidad del modelo (C)** y **el tamaño del conjunto de datos (N)**.

1. Complejidad del Modelo (C)

La complejidad del modelo se refiere al conjunto de funciones o hipótesis que el modelo puede explorar durante el entrenamiento. Por ejemplo:

- Una regresión lineal tiene baja complejidad, ya que solo considera relaciones lineales.
- Una red neuronal profunda tiene alta complejidad debido a su capacidad para representar relaciones no lineales complejas.

Efectos de la Complejidad:

- **Ventajas:** Modelos más complejos pueden capturar patrones intrincados en los datos.
- **Desventajas:** Si el modelo es demasiado complejo para el tamaño de los datos disponibles, puede sobreajustarse, ajustándose incluso al ruido, lo que resulta en un alto E_{out}.

Ejemplo:

Imagina que entrenamos un modelo polinómico para predecir precios de viviendas.

- Un modelo de grado 1 (lineal) podría subajustarse, pasando por alto relaciones importantes.
- Un modelo de grado 20 podría sobreajustarse, capturando el ruido del conjunto de datos.

2. Número de Muestras (N)

El tamaño del conjunto de datos afecta directamente la capacidad del modelo para generalizar. Según la teoría PAC, más datos reducen la brecha entre E_{in} y E_{out}.

Efectos del Tamaño de la Muestra:

- **Ventajas:** Aumentar N mejora la estimación de la distribución subyacente, permitiendo que el modelo capte patrones reales en lugar de ruido.
- **Desventajas:** Recolectar grandes cantidades de datos puede ser costoso o impráctico, y más datos no garantizan mejores resultados si el modelo no es adecuado.

Ejemplo:

- Con 50 muestras, un modelo podría sobreajustarse fácilmente.
- Con 5,000 muestras, el modelo puede identificar patrones consistentes, reduciendo E_{out}.

Garantías Probabilísticas en el Aprendizaje PAC

El aprendizaje PAC no solo caracteriza la brecha de generalización, sino que también ofrece límites probabilísticos para garantizar que esta brecha sea razonable. Dados ϵ y δ como márgenes de error tolerables, el aprendizaje PAC asegura que con al menos una probabilidad $(1-\delta)$:

$$\left|E_{\text{out}} - E_{\text{in}}\right| \leq \epsilon .$$

Interpretación:

- ϵ representa la tolerancia máxima para la diferencia entre los errores.
- δ indica la probabilidad de que esta garantía no se cumpla.

Implicaciones Prácticas del Aprendizaje PAC

El marco PAC proporciona orientación práctica para el diseño y la evaluación de modelos en machine learning:

1. **Compromiso Entre Complejidad y Generalización**

- Modelos simples (bajo C) tienen una menor probabilidad de sobreajustarse, pero pueden subajustarse.
- Modelos complejos (alto C) pueden ajustarse mejor a los datos, pero deben regularse cuidadosamente para evitar el sobreajuste.

2. **Importancia del Tamaño de los Datos**

- En general, un mayor N reduce E_{out}, pero hay un punto de rendimiento decreciente donde más datos no mejoran significativamente el desempeño.

3. **Necesidad de Regularización**
   - Técnicas como L1/L2, dropout, y early stopping son esenciales para manejar la complejidad y evitar el sobreajuste.

4. **Uso Estratégico de Datos**
   - En escenarios donde los datos son limitados, estrategias como el aprendizaje por transferencia y el aumento de datos pueden ser esenciales para cerrar la brecha de generalización.

Ejemplo Práctico: Aprendizaje PAC en Acción

Imagina que estamos entrenando un clasificador para detectar correos electrónicos de spam. Nuestro modelo tiene un espacio de hipótesis con $C = 10^5$, y usamos un conjunto de datos con $N = 1,000$ muestras. Según el aprendizaje PAC:

$$E_{\text{out}} \leq E_{\text{in}} + \sqrt{\frac{\log 10^5}{1,000}}.$$

Calculamos:

- $\log 10^5 = 5 \times \log 10 = 5 \times 2.302 = 11.51$.
- $\sqrt{\frac{11.51}{1,000}} = \sqrt{0.01151} \approx 0.107$.

Esto sugiere que la brecha Ω es de aproximadamente 0.107. Si $E_{\text{in}} = 0.05$, podemos esperar que $E_{\text{out}} \leq 0.157$ con una alta probabilidad.

El aprendizaje PAC ofrece una perspectiva sólida y matemáticamente fundamentada para analizar y mejorar el rendimiento de los modelos de machine learning. Al entender los compromisos entre la complejidad del modelo y el tamaño de los datos, podemos tomar decisiones más informadas que maximicen la capacidad de generalización y minimicen el error en datos no vistos.

PREGUNTAS:

¿Qué sucede si aumentamos el tamaño de muestra N significativamente?

- **Reducción del Error de Generalización:** Un mayor tamaño de muestra, N, permite al modelo aproximarse mejor a la distribución subyacente de los datos, reduciendo la brecha entre el error de entrenamiento (E_{in}) y el error fuera de la muestra (E_{out}). Esto se debe a que el término $\sqrt{\frac{\log C}{N}}$ decrece con el aumento de N.
- **Menor Varianza:** Con más datos, el modelo se vuelve menos sensible a las fluctuaciones del conjunto de entrenamiento, lo que mejora la estabilidad de sus predicciones.
- **Diminución de Beneficios Marginales:** Aunque más datos generalmente mejoran el modelo, llega un punto de rendimiento decreciente en el que los beneficios adicionales son mínimos, especialmente si el modelo ya es adecuado para la complejidad del problema.

Ejemplo:

Si entrenamos un modelo de clasificación con 1,000 muestras y aumentamos a 10,000, la brecha Ω entre E_{in} y E_{out} disminuye aproximadamente en un factor de $\sqrt{10} \approx 3.16$, mejorando la capacidad del modelo para generalizar.

¿Cómo afecta la elección de un modelo con menor complejidad al equilibrio entre E_{in} y E_{out}?

- **Reducción del Riesgo de Sobreajuste:** Un modelo con menor complejidad tiene un espacio de hipótesis más pequeño, lo que reduce la probabilidad de capturar ruido en los datos de entrenamiento, llevando a un menor error fuera de muestra (E_{out}).
- **Aumento del Riesgo de Subajuste:** Si la complejidad es demasiado baja para capturar los patrones subyacentes, el modelo puede tener un alto error de entrenamiento (E_{in}), lo que limita su rendimiento en datos no vistos.
- **Balance Crítico:** La clave es encontrar un modelo con la complejidad justa para ajustarse a los patrones del problema sin sobreajustarse al ruido.

Ejemplo:

- Un modelo de regresión lineal simple (baja complejidad) puede no capturar relaciones no lineales, resultando en alto E_{in}.
- Una red neuronal con regularización adecuada puede balancear E_{in} y E_{out}.

¿Qué estrategias podrían aplicarse cuando los datos disponibles son limitados?

1. **Aumento de Datos (Data Augmentation):**

- Generar ejemplos sintéticos mediante transformaciones de los datos originales, como rotaciones, escalados o adiciones de ruido.
- Útil en problemas como clasificación de imágenes o procesamiento de lenguaje natural.

2. **Aprendizaje por Transferencia (Transfer Learning):**

- Utilizar un modelo preentrenado en un conjunto de datos grande y ajustarlo (fine-tuning) al problema específico con datos limitados.
- Común en tareas de visión por computadora y NLP.

3. **Regularización:**

- Aplicar técnicas como L1/L2, dropout o early stopping para evitar que el modelo se sobreajuste al conjunto reducido de datos.

4. **Modelos con Baja Complejidad:**

- Preferir modelos más simples como regresión lineal o árboles de decisión con restricciones, ya que son menos propensos a sobreajustarse cuando los datos son limitados.

5. **Validación Cruzada (Cross-Validation):**
   - Maximizar el uso de datos disponibles dividiéndolos en múltiples subconjuntos para entrenamiento y evaluación, obteniendo estimaciones más robustas del desempeño del modelo.

6. **Generación de Datos Sintéticos:**
   - Simular datos adicionales mediante técnicas como redes generativas adversarias (GANs) o muestreo a partir de distribuciones estadísticas.

Ejemplo:

En un problema de clasificación de textos con solo 500 ejemplos, se podrían usar embeddings preentrenados como Word2Vec o BERT para reducir la dependencia de un conjunto de datos grande, mientras se aplica regularización para prevenir el sobreajuste.

Aumentar N mejora la generalización, mientras que elegir un modelo con menor complejidad puede reducir el sobreajuste, aunque con riesgo de subajuste. En casos con datos limitados, estrategias como el aumento de datos, transferencia de aprendizaje y regularización son cruciales para optimizar el desempeño del modelo en términos de E_{in} y E_{out}.

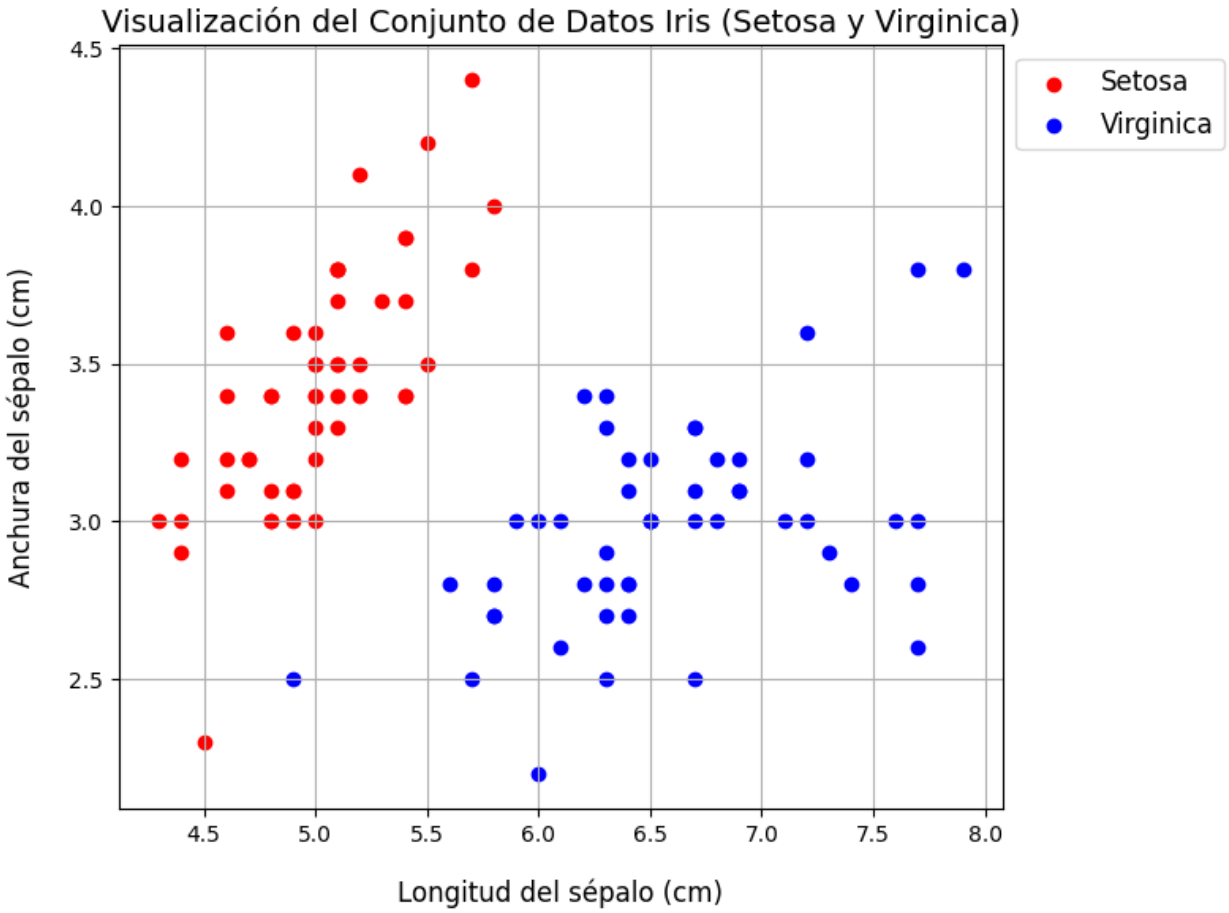
Antes de profundizar en este asunto, abramos el melón del proceso de aprendizaje y observemos de qué partes se compone.

6. Dentro del proceso de aprendizaje

Consideramos un problema de clasificación utilizando el conjunto de datos Iris. Este conjunto contiene muestras de 150 flores, con 4 características (longitud y anchura del sépalo y pétalo) y tres clases: Setosa, Versicolor y Virginica. Para simplificar el ejemplo, eliminaremos la clase **Versicolor** y utilizaremos solo las dos primeras características (longitud del sépalo y anchura del sépalo) para visualizar los datos en dos dimensiones.

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

# Cargar el conjunto de datos Iris
iris = datasets.load_iris()
X = iris.data[iris.target != 1, :2]  # Seleccionar solo dos
características (longitud y anchura del sépalo) y excluir la clase
Versicolor
y = iris.target[iris.target != 1]    # Excluir la clase Versicolor

# Visualizar los datos de las dos clases restantes (Setosa y
Virginica)
plt.figure(figsize=(8, 6))  # Configurar el tamaño de la figura
plt.scatter(X[0:50, 0], X[0:50, 1], color='r', label='Setosa')  #
Puntos rojos para la clase Setosa
```

```
plt.scatter(X[50:, 0], X[50:, 1], color='b', label='Virginica')  # Puntos azules para la clase Virginica

# Añadir título y etiquetas a los ejes
plt.title('Visualización del Conjunto de Datos Iris (Setosa y Virginica)', fontsize=14)  # Título
plt.xlabel('Longitud del sépalo (cm)', fontsize=12, labelpad=15)  # Etiqueta del eje X
plt.ylabel('Anchura del sépalo (cm)', fontsize=12, labelpad=15)  # Etiqueta del eje Y

# Añadir leyenda fuera de la gráfica
plt.legend(loc='upper left', fontsize=12, bbox_to_anchor=(1, 1))

# Activar la cuadrícula
plt.grid(True)

# Mostrar la gráfica
plt.tight_layout()  # Ajustar la distribución para evitar que las etiquetas se corten
plt.show()
```



Para poder aprender, cualquier algoritmo debe definir al menos tres componentes:

- **La clase de modelo/espacio de hipótesis** define la familia de modelos matemáticos que se utilizarán. El límite de decisión objetivo se aproximará a partir de un elemento de este espacio. Por ejemplo, podemos considerar la clase de modelos lineales. En este caso, nuestro límite de decisión será una línea si el problema se define en ${\bf R}^2$ y la clase de modelo es el espacio de todas las líneas posibles en ${\bf R}^2$.

  Las clases de modelos definen las propiedades geométricas de la función de decisión. Existen diferentes taxonomías, pero las más conocidas son las *familias* de modelos **lineales** y **no lineales**. Estas familias generalmente dependen de algunos parámetros. La solución a un problema de aprendizaje es la selección de un conjunto particular de parámetros, es decir, la selección de un modelo dentro del espacio de clases de modelos. Este espacio también se denomina **espacio de hipótesis**.

  Las *familias de modelos lineales* incluyen modelos como la regresión lineal, la regresión logística y las máquinas de soporte vectorial lineales. Estos modelos aproximan el límite de decisión mediante una función lineal, que es fácil de entender e interpretar. Los modelos lineales son efectivos cuando los datos siguen una estructura lineal o casi lineal, pero no son adecuados para problemas más complejos con patrones no lineales.

  Las *familias de modelos no lineales* incluyen modelos como redes neuronales, máquinas de soporte vectorial no lineales y árboles de decisión. Estos modelos permiten aproximaciones de límites de decisión más complejos y pueden manejar patrones de datos más complicados, pero suelen ser más difíciles de interpretar y entrenar. En problemas donde los límites de decisión son altamente complejos o no lineales, los modelos no lineales suelen ser una mejor opción, aunque con el costo de una mayor complejidad computacional.

  La selección del mejor modelo dependerá de nuestro problema y de lo que queramos obtener de él. El objetivo principal en el aprendizaje es lograr el mínimo error o el máximo rendimiento. Sin embargo, dependiendo de lo que se desee del algoritmo, se deben considerar otros factores como la interpretabilidad, el comportamiento frente a datos faltantes, el tiempo de entrenamiento, etc. Además, es fundamental tener en cuenta la capacidad de generalización y la resistencia al sobreajuste.

- **El modelo del problema** formaliza y codifica las propiedades deseadas de la solución. En muchos casos, esta formalización toma la forma de un problema de optimización. En su instanciación más básica, el modelo del problema puede ser la **minimización de una función de error**. Esta función mide la diferencia entre nuestro modelo y el modelo objetivo. Informalmente, en un problema de clasificación, la función de error ideal es la **pérdida 0-1**. Esta función toma el valor 1 cuando clasificamos incorrectamente una muestra de entrenamiento y cero en caso

contrario. En este caso, se puede interpretar como que estamos "irritados" por "una unidad de irritación" cuando una muestra está mal clasificada.

Sin embargo, la **pérdida 0-1** no es diferenciable, lo que dificulta su uso en métodos de optimización basados en gradientes. Por esta razón, se suelen utilizar funciones de error más suaves, como la **entropía cruzada** o la **pérdida cuadrática media** (para regresión). Estas funciones permiten la optimización mediante métodos de gradiente, lo que facilita la actualización de los parámetros del modelo.

El modelo del problema también puede usarse para imponer otras restricciones en nuestra solución, como encontrar una aproximación suave, un modelo de baja complejidad, o una solución dispersa. Estas restricciones son útiles para controlar la complejidad del modelo, evitar el sobreajuste y garantizar que el modelo sea generalizable.

Ejemplos comunes de restricciones incluyen la regularización, que penaliza modelos complejos, y el uso de *dropout* en redes neuronales para prevenir el sobreajuste. La regularización puede tomar varias formas, como la **regularización L1** (que favorece modelos más dispersos) o la **regularización L2** (que favorece modelos con coeficientes pequeños). La regularización también ayuda a prevenir el sobreajuste, lo que mejora la capacidad de generalización del modelo a nuevos datos.

- **El algoritmo de aprendizaje** es un método de optimización o búsqueda que ajusta el modelo a los datos de entrenamiento según la función de error. Existen muchos algoritmos diferentes, dependiendo de la naturaleza del problema. En general, el objetivo es encontrar la aproximación de error mínimo o el modelo más probable máximo. Si el problema es convexo o cuasi-convexo, típicamente se utilizan métodos de primer o segundo orden (por ejemplo, descenso de gradiente, descenso por coordenadas, método de Newton, métodos de punto interior, etc.). Si no se tiene acceso a las derivadas de la función objetivo, se pueden utilizar técnicas como algoritmos genéticos o métodos de Monte Carlo.

  El **descenso de gradiente** es uno de los algoritmos de optimización más populares y se usa ampliamente para entrenar redes neuronales. Este algoritmo ajusta los parámetros del modelo en la dirección del gradiente negativo de la función de error, con el objetivo de minimizarla. Sin embargo, uno de sus principales problemas es que puede ser sensible a la tasa de aprendizaje, lo que puede llevar a una convergencia más lenta o incluso a una divergencia si no se ajusta correctamente.

  El **descenso de gradiente estocástico** (SGD) es una variante del descenso de gradiente en la que, en lugar de calcular el gradiente sobre todo el conjunto de datos, se calcula sobre un subconjunto aleatorio (mini-lote) de los datos, lo que permite acelerar la optimización y manejar grandes volúmenes de datos. Aunque puede ser más ruidoso que el descenso de gradiente tradicional, a menudo converge más rápido.

  En problemas más complejos, como los que involucran datos no estructurados (imágenes o texto), se utilizan **algoritmos de redes neuronales profundas**. Estos

algoritmos permiten aprender representaciones jerárquicas de los datos, pero requieren mayor capacidad computacional y tiempo de entrenamiento.

Otros métodos de optimización incluyen el **algoritmo de Newton** o técnicas de **optimización de segundo orden**, que, aunque más complejas, son más eficientes en problemas convexos al usar la segunda derivada para realizar actualizaciones más precisas.

Vamos a utilizar el "algoritmo de machine learning humano". Mueve los parámetros hasta que sientas que la solución es correcta.

```
# Asegurar que las gráficas se muestren en línea en el Jupyter
Notebook
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from ipywidgets import interact

# Generación de Datos del Conjunto Iris
iris = datasets.load_iris()
X = iris.data[iris.target != 1, :2]  # Seleccionar solo las dos
primeras características y excluir Versicolor
y = iris.target[iris.target != 1]    # Excluir la clase Versicolor

# Función para mostrar la frontera de decisión interactiva
def decision_boundary(w0, w1, offset):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
np.arange(y_min, y_max, 0.1))
    Z = np.sign(w0 * xx + w1 * yy + offset)  # Evaluar la frontera de
decisión
    plt.figure(figsize=(8, 6))  # Configurar el tamaño de la figura
    plt.contourf(xx, yy, Z, alpha=0.4)  # Mostrar la frontera

    # Usar diferentes colores para las clases Setosa y Virginica
    scatter = plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k',
cmap=plt.cm.RdYlBu)  # Mostrar los puntos de datos
    plt.xlabel('Longitud del sépalo (cm)', fontsize=12)  # Etiqueta
del eje X
    plt.ylabel('Anchura del sépalo (cm)', fontsize=12)  # Etiqueta del
eje Y
    plt.title('Frontera de decisión - Algoritmo de Machine Learning
Humano', fontsize=14)  # Título

    # Etiquetas para las clases Setosa y Virginica
    handles, labels = scatter.legend_elements()
    plt.legend(handles, ['Setosa', 'Virginica'], loc='upper left',
```

```
fontsize=12, bbox_to_anchor=(1, 1))

    plt.show()

# Crear la interfaz interactiva para ajustar los parámetros
interact(decision_boundary, w0=(-2.0, 2.0, 0.1), w1=(-2.0, 2.0, 0.1),
offset=(-2.0, 2.0, 0.1))

# Solución (w0, w1, offset) = (0.8,-1,-1.4)
```

{"model_id":"b9a4a90d24f5453ca84bdfd76fc566b1","version_major":2,"version_minor":0}

```
<function __main__.decision_boundary(w0, w1, offset)>
```

El **algoritmo de machine learning humano** representa un proceso iterativo de ajuste. A diferencia de los algoritmos tradicionales de optimización matemática, este proceso no sigue una lógica estricta y precisa, sino que depende de la intuición y juicio del ser humano para ajustar los parámetros y encontrar una solución adecuada. Este enfoque es más flexible, aunque puede ser menos eficiente en términos de convergencia a una solución óptima.

PREGUNTA: ¿Cómo se describiría paso a paso el proceso utilizado para ajustar el clasificador?

El proceso de ajuste del clasificador en este ejemplo es interactivo y se basa en un enfoque manual para ajustar parámetros. A continuación se describe paso a paso cómo se lleva a cabo este proceso:

1. **Generación de Datos del Conjunto Iris:** Se cargan los datos del conjunto Iris, seleccionando solo las clases Setosa y Virginica, y utilizando únicamente las características de longitud y anchura del sépalo.

2. **Visualización de los Datos:** Los puntos generados se visualizan en un gráfico bidimensional. Los puntos de la clase Setosa se representan en rojo, mientras que los de la clase Virginica se muestran en azul. Esta visualización permite ver cómo se distribuyen los puntos y cómo se puede observar la separación entre las clases.

3. **Definición de la Función de Frontera de Decisión:** Se crea una malla de puntos que cubre todo el plano bidimensional. Luego, cada punto de esta malla se evalúa usando los parámetros del modelo (pesos y offset) para determinar a qué clase pertenece. La frontera de decisión se define como el conjunto de puntos donde el modelo no puede clasificar claramente, es decir, donde la salida es cero.

4. **Interacción del Usuario:** Se proporcionan controles deslizantes (widgets interactivos) para que el usuario ajuste manualmente los parámetros del modelo. Estos parámetros incluyen los pesos w_0 y w_1, que definen la pendiente de la línea de decisión, y el offset, que ajusta su desplazamiento. A medida que el usuario modifica estos parámetros, la frontera de decisión se actualiza en tiempo real, lo que permite observar cómo varía la clasificación de los puntos.

5. **Ajuste de los Parámetros:** El objetivo del usuario es mover los controles deslizantes para que la frontera de decisión se alinee lo mejor posible con la separación natural entre las clases. Este ajuste manual emula el proceso de entrenamiento de un clasificador lineal. Si el proceso fuera automatizado, un algoritmo de optimización ajustaría los parámetros para minimizar el error de clasificación, buscando la mejor separación posible entre las clases.

6. **Visualización del Modelo Ajustado:** Una vez que el usuario ha ajustado los parámetros para que la frontera de decisión se alinee de manera efectiva con la separación de las clases, la configuración final de la frontera se muestra en el gráfico. Esto proporciona una representación visual clara de cómo el clasificador divide el espacio entre las dos clases.

Algunas notas sobre el proceso de aprendizaje

El objetivo principal de cualquier proceso de aprendizaje es maximizar el poder predictivo (*precisión*) del modelo, lo que implica minimizar el error de clasificación. Sin embargo, existen otras propiedades deseables que también deben ser consideradas al desarrollar modelos de machine learning:

- **Simplicidad** - ¿Qué tan simple es el modelo? ¿Cuánto ajuste es necesario para que funcione correctamente? ¿Se puede modificar el modelo para adaptarlo a las particularidades de un problema específico?
- **Velocidad** - ¿Cuánto tiempo toma entrenar un modelo confiable? (tiempo de entrenamiento) ¿Es capaz de hacer predicciones rápidas, como en aplicaciones en tiempo real? (tiempo de prueba) ¿Cuánto tiempo sería necesario para procesar un conjunto de datos extremadamente grande, como un yottabyte (1e24 Bytes)?
- **Interpretabilidad** - ¿Por qué el modelo realiza una predicción específica? ¿Es fácil entender cómo se llegó a una decisión o resultado?

Es importante destacar que a menudo la precisión se debe balancear con otras propiedades, ya que la mejora de una puede comprometer las demás. Encontrar el equilibrio adecuado es esencial para desarrollar modelos de machine learning efectivos y eficientes.

7. Curvas de aprendizaje y sobreajuste (Overfitting)

Volvamos a la capacidad de aprendizaje PAC.

```
# Asegurar que las gráficas se muestren en línea en el Jupyter
Notebook
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, neighbors
```

```
from ipywidgets import interact

# Generación de Datos del Conjunto Iris
iris = datasets.load_iris()
X = iris.data[iris.target != 1, :2]  # Seleccionar solo las dos
primeras características y excluir Versicolor
y = iris.target[iris.target != 1]    # Excluir la clase Versicolor
y = np.where(y == 0, -1, 1)  # Cambiar etiquetas de Setosa a -1 y
Virginica a 1

# Definir constantes MAXC y MAXN
MAXC = 50  # Número máximo de vecinos para KNN
MAXN = len(X)  # Número máximo de muestras para entrenamiento
(longitud real de X)

# Función para mostrar la frontera de decisión interactiva
def decision_boundary_knn(C, N):
    # Seleccionar un subconjunto de N muestras
    Xr = X[:N, :]
    yr = y[:N]

    # Crear malla para evaluar la frontera de decisión
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
np.arange(y_min, y_max, 0.1))
    X_mesh = np.c_[xx.ravel(), yy.ravel()]

    # Crear y entrenar el clasificador KNeighborsClassifier
    clf = neighbors.KNeighborsClassifier(n_neighbors=C)
    clf.fit(Xr, yr)

    # Predecir en la malla
    Z = clf.predict(X_mesh)
    Z = Z.reshape(xx.shape)

    # Visualización
    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.RdYlBu)  # Mostrar
la frontera

    # Usar diferentes colores para las clases Setosa y Virginica
    scatter = plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k',
cmap=plt.cm.RdYlBu)
    plt.xlabel('Longitud del sépalo (cm)', fontsize=12)
    plt.ylabel('Anchura del sépalo (cm)', fontsize=12)
    plt.title('Frontera de decisión con KNN - Iris Dataset',
fontsize=14)

    # Etiquetas para las clases Setosa y Virginica
```

```
    handles, labels = scatter.legend_elements()
    plt.legend(handles, ['Setosa', 'Virginica'], loc='upper left',
fontsize=12, bbox_to_anchor=(1, 1))

    plt.show()

# Crear la interfaz interactiva para ajustar los parámetros
interact(decision_boundary_knn, C=(1, MAXC), N=(20, MAXN))

# Solución (C, N) = (1,57)
```

{"model_id":"639a98853689435c98b90563ce86c64f","version_major":2,"version_minor":0}

```
<function __main__.decision_boundary_knn(C, N)>
```

EJERCICIO:

1. **Establece el número de muestras de datos por grupo N en 100 y el valor de complejidad C en 50. Describe lo que observas: ¿El método clasifica incorrectamente alguna muestra de datos?**

   - **Descripción de lo que observas**: Cuando el número de muestras es 100 y la complejidad se establece en $C=50$ (es decir, el clasificador se ajusta con un número elevado de vecinos), el clasificador KNN tiende a ser muy flexible y ajustado a los datos de entrenamiento. La frontera de decisión se adapta de forma muy precisa a las muestras presentes, lo que puede llevar a un sobreajuste (overfitting). Esto significa que el modelo podría clasificar correctamente la mayoría de las muestras de entrenamiento, pero podría ser susceptible a clasificar incorrectamente muestras fuera del conjunto de entrenamiento.

     ¿El método clasifica incorrectamente alguna muestra de datos? Con $C=50$ y $N=100$, es posible que el modelo no cometa muchos errores en el conjunto de entrenamiento, pero podría haber ciertos puntos cerca de la frontera de decisión que podrían ser clasificados erróneamente, especialmente si los datos de prueba no siguen el mismo patrón exacto que los datos de entrenamiento.

2. **Disminuye el valor de complejidad a $C=20$. Describe el límite: ¿El método clasifica incorrectamente alguna muestra de datos?**

   - **Descripción del límite**: Al reducir la complejidad (número de vecinos a considerar en la clasificación), la frontera de decisión se vuelve más suave y menos ajustada a las muestras de entrenamiento. Esto ayuda a evitar el sobreajuste. Ahora, con $C=20$, el modelo será menos sensible a pequeñas variaciones en los datos, lo que puede resultar en una mejor generalización frente a datos nuevos.

¿El método clasifica incorrectamente alguna muestra de datos? Al reducir la complejidad, es probable que el modelo cometa algunos errores, especialmente en las muestras cercanas a la frontera entre las dos clases (Setosa y Virginica). El número de errores puede ser mayor en comparación con la configuración con $C=50$, ya que ahora el modelo no está tan ajustado a los puntos de entrenamiento, pero es más probable que generalice mejor en datos fuera de muestra.

3. **¿Cuál de las dos configuraciones crees que funcionará mejor frente a nuevos datos de la misma distribución?**

   - **Respuesta**: La configuración con $C=20$ probablemente funcionará mejor frente a nuevos datos de la misma distribución. Esto se debe a que una complejidad menor (menos vecinos) ayuda a evitar el sobreajuste, lo que significa que el modelo tendrá una frontera de decisión más general y menos sensible a ruidos o peculiaridades de los datos de entrenamiento. Si bien podría cometer más errores en los datos de entrenamiento, es más probable que generalice bien y mantenga un buen rendimiento en nuevos datos.

     En cambio, con $C=50$, el modelo podría estar muy ajustado a los datos de entrenamiento, lo que puede llevar a un rendimiento subóptimo cuando se enfrente a nuevos puntos de datos que no sigan la misma distribución exacta.

7.1 Curvas de aprendizaje

Visualicemos el comportamiento observado. Para este propósito, podemos trazar una curva del error de entrenamiento y del error de prueba a medida que aumenta el número de datos de entrenamiento para una complejidad dada. Esta curva se llama **curva de aprendizaje**.

Las curvas de aprendizaje son herramientas fundamentales en el campo de machine learning, ya que proporcionan información valiosa sobre cómo un modelo de predicción mejora o se deteriora conforme se le proporcionan diferentes cantidades de datos de entrenamiento. Al observar estas curvas, podemos diagnosticar problemas como el sobreajuste (overfitting) o el subajuste (underfitting), que son fundamentales para mejorar el rendimiento del modelo y su capacidad de generalización.

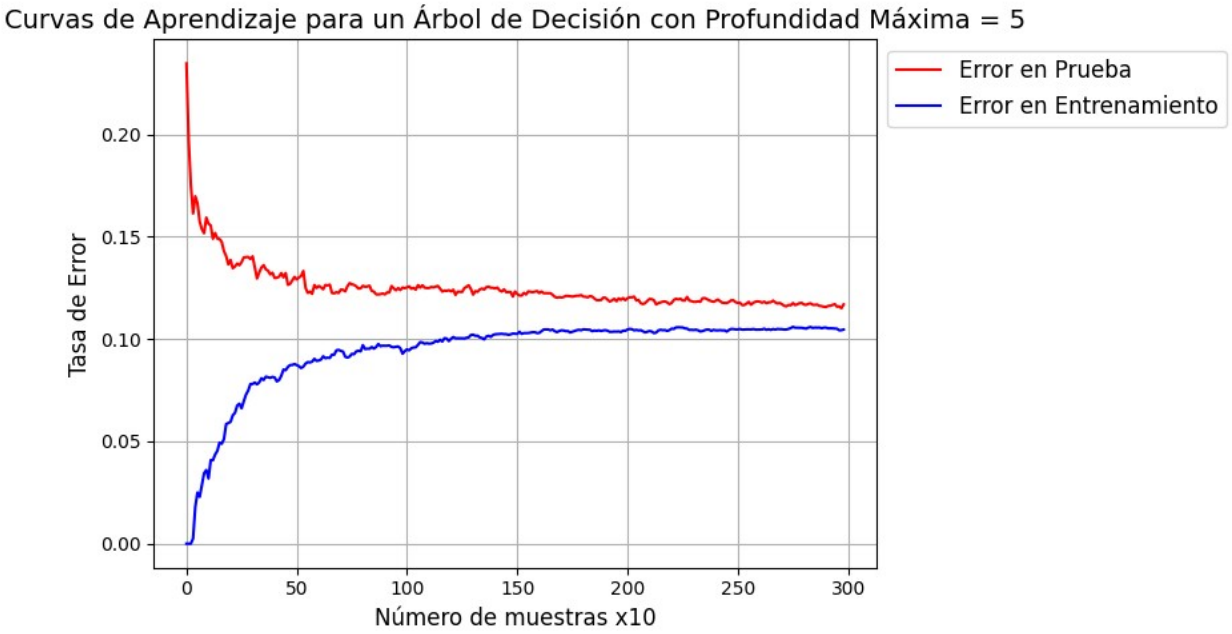
Las curvas de aprendizaje ayudan a entender cómo cambia la tasa de error a medida que varían los tamaños de los conjuntos de entrenamiento y prueba. Este análisis es crucial para saber si un modelo está aprendiendo correctamente o si es necesario ajustar algunos parámetros para obtener un mejor desempeño.

```
# Asegurar que las gráficas se muestren en línea en el Jupyter
Notebook
%matplotlib inline

# Importar las bibliotecas necesarias.
import numpy as np  # Para operaciones numéricas.
import matplotlib.pyplot as plt  # Para graficar gráficos.
from sklearn import metrics  # Para calcular métricas de rendimiento
```

```
del modelo.
from sklearn import tree  # Para utilizar modelos de árbol de
decisión.

# Establecer la semilla para la aleatorización (asegura resultados
repetibles).
np.random.seed(42)

# Establecer la complejidad del árbol de decisión en 5 (profundidad
máxima de 5).
C = 5  # Profundidad máxima del árbol de decisión (establecido en 5).
MAXN = 1000  # Número máximo de muestras por clase.

# Inicializar matrices para almacenar las tasas de error para 10
iteraciones y diferentes números de muestras de entrenamiento.
yhat_test_c5 = np.zeros((10, 299, 2))  # Error en prueba (tamaño:
[iteraciones, tamaños de muestra, conjunto])
yhat_train_c5 = np.zeros((10, 299, 2))  # Error en entrenamiento

# Ejecutar el experimento 10 veces para obtener curvas suavizadas
promedio.
for iteration in range(10):
    # Generar datos sintéticos para el conjunto de entrenamiento.
    # Usar distribuciones normales para crear datos representativos.
    X_train = np.concatenate([
        1.25 * np.random.randn(MAXN, 2),  # Primera distribución
        5 + 1.5 * np.random.randn(MAXN, 2),  # Segunda distribución
        [8, 5] + 1.5 * np.random.randn(MAXN, 2)  # Tercera
distribución
    ])
    y_train = np.concatenate([
        np.ones((MAXN, 1)),  # Etiquetas para la primera clase
        -np.ones((MAXN, 1)),  # Etiquetas para la segunda clase
        np.ones((MAXN, 1))  # Etiquetas para la tercera clase
    ])

    # Aleatorizar los datos para asegurar la variabilidad.
    perm = np.random.permutation(y_train.size)
    X_train = X_train[perm, :]
    y_train = y_train[perm]

    # Generar datos sintéticos para el conjunto de prueba.
    X_test = np.concatenate([
        1.25 * np.random.randn(MAXN, 2),
        5 + 1.5 * np.random.randn(MAXN, 2),
        [8, 5] + 1.5 * np.random.randn(MAXN, 2)
    ])
    y_test = np.concatenate([
        np.ones((MAXN, 1)),
        -np.ones((MAXN, 1)),
```

```
        np.ones((MAXN, 1))
    ])

    # Evaluar el modelo con diferentes tamaños de muestra.
    for j, N in enumerate(range(10, 3000, 10)):
        # Tomar un subconjunto de datos para entrenamiento con los
primeros N ejemplos.
        X_subset = X_train[:N, :]
        y_subset = y_train[:N]

        # Crear y entrenar un clasificador de árbol de decisión con
una profundidad máxima de C (ahora 5).
        clf = tree.DecisionTreeClassifier(min_samples_leaf=1,
max_depth=C)
        clf.fit(X_subset, y_subset.ravel())  # Ajustar el modelo a los
datos de entrenamiento.

        # Evaluar el modelo en el conjunto de prueba y calcular la
tasa de error.
        yhat_test_c5[iteration, j, 1] = 1 -
metrics.accuracy_score(clf.predict(X_test), y_test.ravel())
        # Evaluar el modelo en el conjunto de entrenamiento y calcular
la tasa de error.
        yhat_train_c5[iteration, j, 1] = 1 -
metrics.accuracy_score(clf.predict(X_subset), y_subset.ravel())

# Promediar las tasas de error a lo largo de las iteraciones para
obtener una curva de aprendizaje más estable.
mean_test_error_c5 = np.mean(yhat_test_c5[:, :, 1].T, axis=1)
mean_train_error_c5 = np.mean(yhat_train_c5[:, :, 1].T, axis=1)

# Graficar las curvas de aprendizaje: tasa de error en prueba y
entrenamiento.
plt.figure(figsize=(9, 5))
plt.plot(mean_test_error_c5, 'r', label='Error en Prueba')  # Error en
el conjunto de prueba.
plt.plot(mean_train_error_c5, 'b', label='Error en Entrenamiento')  #
Error en el conjunto de entrenamiento.

# Configurar etiquetas, título y leyenda.
plt.xlabel('Número de muestras x10', fontsize=12)
plt.ylabel('Tasa de Error', fontsize=12)
plt.title('Curvas de Aprendizaje para un Árbol de Decisión con
Profundidad Máxima = 5', fontsize=14)
plt.legend(loc='upper left', fontsize=12, bbox_to_anchor=(1, 1))

# Mostrar la gráfica con las curvas de error.
plt.grid(True)
plt.tight_layout()  # Ajustar la distribución para evitar que las
```

```
etiquetas se corten.
plt.show()
```



Entendiendo las Curvas de Aprendizaje

El gráfico que estamos observando se conoce como "curva de aprendizaje". Representa cómo cambian las tasas de error de un modelo de machine learning a medida que aumenta el tamaño del conjunto de datos de entrenamiento. La línea roja denota el error en el conjunto de prueba (error de generalización), y la línea azul representa el error en el conjunto de entrenamiento (error de entrenamiento).

Observaciones de la Curva de Aprendizaje

- **Convergencia al Sesgo:** A medida que aumenta el número de muestras de entrenamiento, tanto el error de entrenamiento como el de prueba convergen hacia un valor estable. Este valor estable puede interpretarse como el sesgo inherente del modelo, que es la tasa de error que nuestro modelo siempre tendrá, incluso si tuviéramos una cantidad infinita de datos de entrenamiento. El sesgo es el resultado de que las suposiciones del modelo no se alinean perfectamente con las verdaderas relaciones en los datos subyacentes.
  - *Ejemplo de Sesgo:* Si un modelo intenta predecir un valor de una variable de manera lineal (como una regresión lineal) cuando la relación real es no lineal, este modelo sufrirá un alto sesgo. La curva de aprendizaje mostrará una tasa de error de prueba relativamente constante que no mejora a pesar de aumentar el tamaño de los datos de entrenamiento.
- **Sobreajuste con Conjuntos Pequeños de Entrenamiento:** Con una cantidad muy pequeña de datos de entrenamiento, el error de entrenamiento es bastante bajo, lo que podría parecer inicialmente positivo. Sin embargo, el error de prueba

correspondiente es significativamente más alto, lo que indica que el modelo está sobreajustado. El sobreajuste ocurre porque el modelo aprende patrones que son específicos del pequeño conjunto de entrenamiento pero no se generalizan a la distribución más amplia de datos.

  - *Ejemplo de Sobreajuste:* Si un modelo tiene demasiados parámetros en comparación con la cantidad de datos de entrenamiento, puede memorizar los datos de entrenamiento sin aprender las relaciones generales subyacentes. Esto genera un rendimiento excelente en los datos de entrenamiento pero pobre en datos no vistos.

- **Meseta de la Tasa de Error:** Más allá de cierto punto, agregar más muestras de entrenamiento no resulta en mejoras sustanciales en el error de prueba. Esta meseta indica que simplemente agregar más datos no mejorará el rendimiento del modelo y sugiere que hemos alcanzado el límite de lo que el modelo puede aprender dado su capacidad actual.

  - *Ejemplo de la Meseta:* Si el modelo alcanza una tasa de error de prueba que no mejora con más datos, es probable que el modelo haya aprendido la mayoría de las relaciones posibles dentro de los datos disponibles. En este caso, los beneficios de añadir más datos son limitados.

Explicación Teórica

Las curvas de aprendizaje están basadas en la teoría del aprendizaje estadístico. Teóricamente, esperamos que el error de entrenamiento de un modelo aumente y el error de prueba disminuya a medida que crece el número de ejemplos de entrenamiento. Inicialmente, con pocos puntos de datos, un modelo es capaz de ajustarse muy de cerca a los datos de entrenamiento. A medida que se agregan más puntos de datos, se hace más difícil para el modelo ajustar todos los puntos perfectamente, por lo tanto, el error de entrenamiento aumenta. Por el contrario, con más datos, la generalización del modelo a datos no vistos mejora, por lo tanto, el error de prueba disminuye.

Sin embargo, ambos errores convergerán en un punto donde datos adicionales no cambian significativamente las tasas de error. Este punto de convergencia refleja el trade-off de sesgo y varianza inherente del modelo.

- **Sesgo:** Representa el error de suposiciones erróneas en el algoritmo de aprendizaje. Un alto sesgo puede hacer que el modelo no detecte relaciones relevantes entre las características y las salidas objetivo (subajuste).

- **Varianza:** Representa el error de sensibilidad a pequeñas fluctuaciones en el conjunto de entrenamiento. Una alta varianza puede causar sobreajuste: modelar el ruido aleatorio en los datos de entrenamiento, en lugar de las salidas previstas.

Optimizar un modelo implica encontrar el equilibrio adecuado entre sesgo y varianza, lo cual puede analizarse visualmente usando una curva de aprendizaje. Si la curva de aprendizaje ha alcanzado una meseta y todavía hay una brecha significativa entre los errores de entrenamiento y prueba, esto puede sugerir que el modelo tiene alta varianza. En este caso, obtener más datos de entrenamiento probablemente no conducirá a una mejor generalización. En su lugar, se

podría considerar aumentar la complejidad del modelo o utilizar un algoritmo más sofisticado para reducir el sesgo.

El Caso de Modelos No Lineales y Regularización

En el caso de modelos no lineales, como las redes neuronales o los SVM con núcleos complejos, las curvas de aprendizaje pueden ser más difíciles de interpretar. Por ejemplo, en modelos con alta varianza, las curvas de aprendizaje pueden no estabilizarse rápidamente, y es posible que se requiera más datos o técnicas como la regularización para mejorar la generalización. La regularización ayuda a controlar la complejidad del modelo, evitando que se sobreajuste a los datos de entrenamiento.

Más Consideraciones en la Práctica

En la práctica, las curvas de aprendizaje son cruciales para la evaluación de modelos en situaciones reales, donde los conjuntos de datos pueden no ser tan simples como en los ejemplos sintéticos. Cuando se trata de datos más complejos, como imágenes o texto, las curvas de aprendizaje pueden ayudar a decidir si el modelo necesita más datos, ajustes de hiperparámetros o técnicas de regularización.

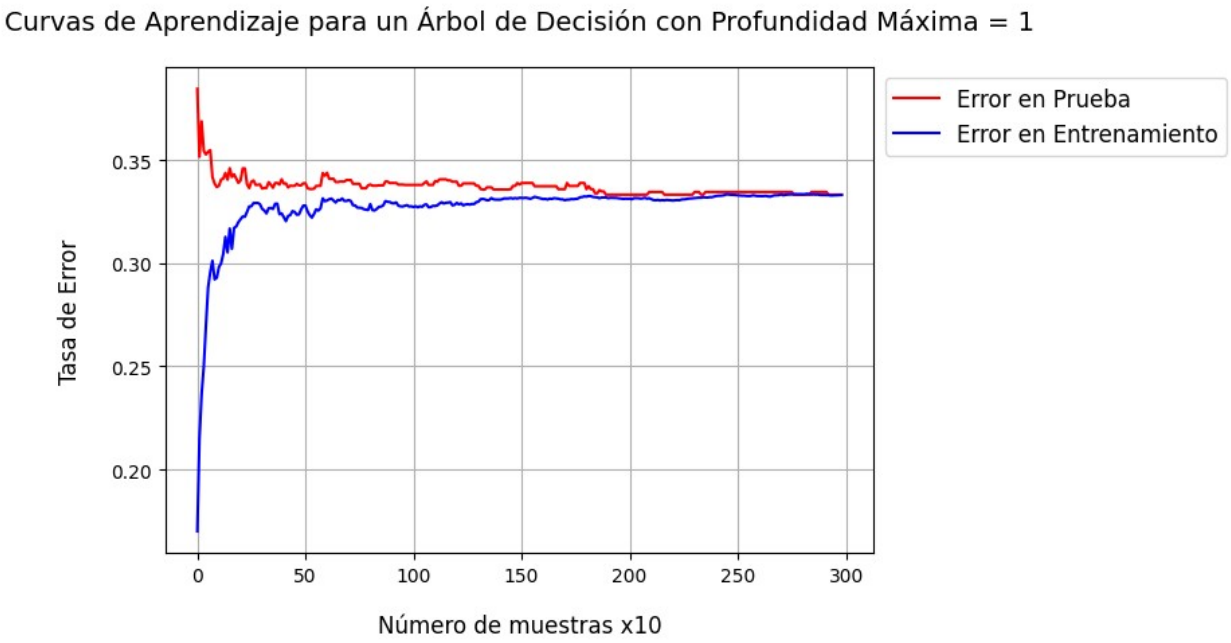
Además, las curvas de aprendizaje también pueden utilizarse para detectar posibles problemas como la selección de características incorrectas o la elección de un modelo inapropiado, proporcionando una guía más específica sobre cómo mejorar el rendimiento.

Finalmente, no hay una "mejor" curva de aprendizaje. Las curvas siempre deben analizarse en el contexto del problema específico y las características del conjunto de datos, lo que hace que su interpretación sea una habilidad esencial en el desarrollo de modelos de machine learning.

```
# Asegurar que las gráficas se muestren en línea en el Jupyter
Notebook
%matplotlib inline

# Importar las bibliotecas necesarias.
import numpy as np  # Para operaciones numéricas.
import matplotlib.pyplot as plt  # Para graficar gráficos.
from sklearn import metrics  # Para calcular métricas de rendimiento
del modelo.
from sklearn import tree  # Para utilizar modelos de árbol de
decisión.

# Establecer la semilla para la aleatorización (asegura resultados
repetibles).
np.random.seed(42)

# Establecer la complejidad del árbol de decisión en 1 (árbol de
decisión con una profundidad máxima de 1).
C = 1  # Profundidad máxima del árbol de decisión (establecido en 1).
MAXN = 1000  # Número máximo de muestras por clase.

# Inicializar matrices para almacenar las tasas de error para 10
iteraciones y diferentes números de muestras de entrenamiento.
```

```
yhat_test_c1 = np.zeros((10, 299, 2))  # Error en prueba (tamaño:
[iteraciones, tamaños de muestra, conjunto])
yhat_train_c1 = np.zeros((10, 299, 2))  # Error en entrenamiento

# Ejecutar el experimento 10 veces para obtener curvas suavizadas
promedio.
for iteration in range(10):
    # Generar datos sintéticos para el conjunto de entrenamiento.
    # Usar distribuciones normales para crear datos representativos.
    X_train = np.concatenate([
        1.25 * np.random.randn(MAXN, 2),  # Primera distribución
        5 + 1.5 * np.random.randn(MAXN, 2),  # Segunda distribución
        [8, 5] + 1.5 * np.random.randn(MAXN, 2)  # Tercera
distribución
    ])
    y_train = np.concatenate([
        np.ones((MAXN, 1)),  # Etiquetas para la primera clase
        -np.ones((MAXN, 1)),  # Etiquetas para la segunda clase
        np.ones((MAXN, 1))  # Etiquetas para la tercera clase
    ])

    # Aleatorizar los datos para asegurar la variabilidad.
    perm = np.random.permutation(y_train.size)
    X_train = X_train[perm, :]
    y_train = y_train[perm]

    # Generar datos sintéticos para el conjunto de prueba.
    X_test = np.concatenate([
        1.25 * np.random.randn(MAXN, 2),
        5 + 1.5 * np.random.randn(MAXN, 2),
        [8, 5] + 1.5 * np.random.randn(MAXN, 2)
    ])
    y_test = np.concatenate([
        np.ones((MAXN, 1)),
        -np.ones((MAXN, 1)),
        np.ones((MAXN, 1))
    ])

    # Evaluar el modelo con diferentes tamaños de muestra.
    for j, N in enumerate(range(10, 3000, 10)):
        # Tomar un subconjunto de datos para entrenamiento con los
primeros N ejemplos.
        X_subset = X_train[:N, :]
        y_subset = y_train[:N]

        # Crear y entrenar un clasificador de árbol de decisión con
una profundidad máxima de C (ahora 1).
        clf = tree.DecisionTreeClassifier(min_samples_leaf=1,
max_depth=C)
        clf.fit(X_subset, y_subset.ravel())  # Ajustar el modelo a los
```

```
datos de entrenamiento.

        # Evaluar el modelo en el conjunto de prueba y calcular la
tasa de error.
        yhat_test_c1[iteration, j, 1] = 1 -
metrics.accuracy_score(clf.predict(X_test), y_test.ravel())
        # Evaluar el modelo en el conjunto de entrenamiento y calcular
la tasa de error.
        yhat_train_c1[iteration, j, 1] = 1 -
metrics.accuracy_score(clf.predict(X_subset), y_subset.ravel())

# Promediar las tasas de error a lo largo de las iteraciones para
obtener una curva de aprendizaje más estable.
mean_test_error_c1 = np.mean(yhat_test_c1[:, :, 1].T, axis=1)
mean_train_error_c1 = np.mean(yhat_train_c1[:, :, 1].T, axis=1)

# Graficar las curvas de aprendizaje: tasa de error en prueba y
entrenamiento.
plt.figure(figsize=(9, 5))
plt.plot(mean_test_error_c1, 'r', label='Error en Prueba')  # Error en
el conjunto de prueba.
plt.plot(mean_train_error_c1, 'b', label='Error en Entrenamiento')  #
Error en el conjunto de entrenamiento.

# Configurar etiquetas, título y leyenda.
plt.xlabel('Número de muestras x10', fontsize=12, labelpad=15)  #
Etiqueta del eje X con espacio fuera del gráfico.
plt.ylabel('Tasa de Error', fontsize=12, labelpad=15)  # Etiqueta del
eje Y con espacio fuera del gráfico.
plt.title('Curvas de Aprendizaje para un Árbol de Decisión con
Profundidad Máxima = 1', fontsize=14, pad=20)  # Título con espacio
fuera del gráfico.
plt.legend(loc='upper left', fontsize=12, bbox_to_anchor=(1, 1))

# Mostrar la gráfica con las curvas de error.
plt.grid(True)
plt.tight_layout()  # Ajustar la distribución para evitar que las
etiquetas se corten.
plt.show()
```



Análisis de Curvas de Aprendizaje con Menor Complejidad

En el código proporcionado, repetimos un experimento para trazar curvas de aprendizaje de un clasificador de árbol de decisión, esta vez con una complejidad (`C`) de 1, indicando un modelo más simple con una profundidad máxima de 1. Al iterar este proceso diez veces y promediar los resultados, apuntamos a obtener curvas de aprendizaje suaves que representen el comportamiento promedio del modelo.

Percepciones de Curvas de Aprendizaje con Modelos Más Simples

- **Sesgo Aumentado:** Con `C` configurado en 1, el árbol de decisión es muy simple y probablemente tenga un alto sesgo, lo que significa que hace suposiciones fuertes sobre la forma del límite de decisión. En la práctica, esto puede llevar a un subajuste, donde el modelo es demasiado simplista para capturar la complejidad de los datos.
- **Tasas de Error Convergentes:** Similar al ejemplo anterior, a medida que aumenta el número de muestras de entrenamiento, esperamos que tanto los errores de entrenamiento como de prueba converjan. Sin embargo, debido al sesgo aumentado, pueden converger a una tasa de error más alta en comparación con un modelo más complejo.
- **Meseta de Tasa de Error:** La meseta o nivelación de la tasa de error ocurre a un valor más alto, lo que es indicativo de la capacidad limitada del modelo. Dado que el modelo es simple, podría no beneficiarse tanto de datos de entrenamiento adicionales más allá de cierto punto.

Antecedentes Teóricos

Las curvas de aprendizaje trazadas aquí se basan en la comprensión teórica del impacto de la complejidad del modelo en el aprendizaje. Un modelo más simple con `C=1` generalmente exhibirá las siguientes características:

- **Error de Entrenamiento Más Alto:** Un modelo más simple no se ajustará tan bien a los datos de entrenamiento, lo que lleva a un error de entrenamiento más alto.
- **Varianza Menor:** La brecha entre los errores de entrenamiento y prueba es típicamente más pequeña para modelos más simples, ya que son menos sensibles al ruido específico en los datos de entrenamiento.
- **Meseta Temprana:** Debido a la simplicidad del modelo, las tasas de error alcanzarán una meseta temprano, ya que hay menos capacidad para que el modelo aprenda de datos adicionales.

Implicaciones Prácticas

Cuando la profundidad del árbol de decisión está limitada a un nivel (`C=1`), el límite de decisión es esencialmente una única división basada en una característica. Esta simplicidad puede ser beneficiosa si sospechamos que una característica es predominantemente importante, pero más a menudo, los datos del mundo real son más complejos y un modelo tan simple tendrá un rendimiento inferior.

La curva de aprendizaje probablemente mostrará una tasa de error más alta que se estabiliza rápidamente, reflejando la incapacidad del modelo para reducir aún más el error, independientemente de más datos de entrenamiento. Esta situación sugiere que podemos ver un comportamiento similar en esta segunda curva.

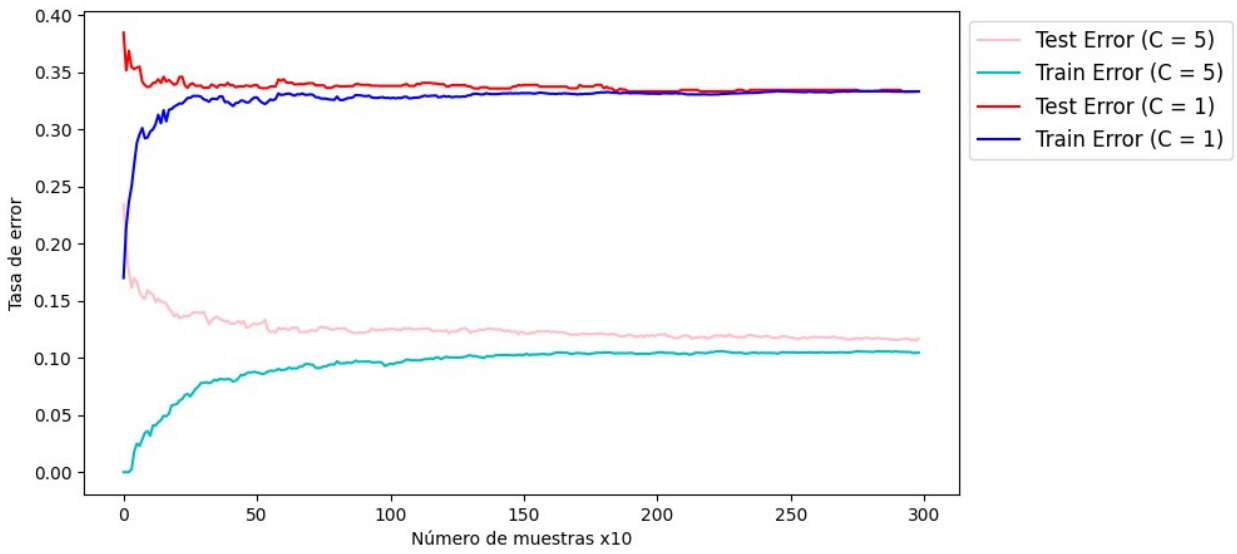
Además de la comparación de los errores de prueba y entrenamiento en modelos de mayor y menor complejidad, es crucial reflexionar sobre las implicaciones prácticas de cada enfoque.

Comparación de Modelos con Diferentes Niveles de Complejidad

A continuación, comparamos las curvas de aprendizaje de dos modelos: uno con mayor complejidad (`C = 5`) y otro con menor complejidad (`C = 1`). La idea es ilustrar cómo la profundidad del árbol afecta la capacidad del modelo para generalizar:

```
# Graficar el error promedio de prueba para el primer conjunto de
experimentos con mayor complejidad (C = 5) en color rosa.
p1, = plt.plot(np.mean(yhat_test_c5[:,:,1].T, axis=1), color='pink',
label='Test Error (C = 5)')

# Graficar el error promedio de entrenamiento para el primer conjunto
de experimentos con mayor complejidad (C = 5) en cian.
p2, = plt.plot(np.mean(yhat_train_c5[:,:,1].T, axis=1), 'c',
label='Train Error (C = 5)')

# Graficar el error promedio de prueba para el segundo conjunto de
experimentos con menor complejidad (C = 1) en rojo.
p3, = plt.plot(np.mean(yhat_test_c1[:,:,1].T, axis=1), 'r',
label='Test Error (C = 1)')
```

```
# Graficar el error promedio de entrenamiento para el segundo conjunto de experimentos con menor complejidad (C = 1) en azul.
p4, = plt.plot(np.mean(yhat_train_c1[:,:,1].T, axis=1), 'b', label='Train Error (C = 1)')

# Configurar el tamaño de la figura para una mejor visibilidad.
fig = plt.gcf()
fig.set_size_inches(9, 5)

# Etiquetar el eje x como 'Número de muestras x10'.
plt.xlabel('Número de muestras x10')

# Etiquetar el eje y como 'Tasa de error'.
plt.ylabel('Tasa de error')

# Agregar una leyenda al gráfico para identificar cada línea.
plt.legend(handles=[p1, p2, p3, p4], labels=["Test Error (C = 5)", "Train Error (C = 5)", "Test Error (C = 1)", "Train Error (C = 1)"])
plt.legend(loc='upper left', fontsize=12, bbox_to_anchor=(1, 1))

# Mostrar el gráfico.
plt.show()
```



En este gráfico se comparan dos modelos de árboles de decisión con diferentes niveles de complejidad, representados por `C = 5` y `C = 1`. Estos valores de `C` indican la profundidad del árbol, lo cual influye directamente en la capacidad del modelo para ajustarse a los datos. Los colores rosa y rojo indican las tasas de error de prueba para los modelos más complejos (`C = 5`) y más simples (`C = 1`), respectivamente, mientras que las líneas azul y cian representan las tasas de error de entrenamiento de los mismos modelos.

Observaciones Clave e Interpretaciones

- **Convergencia Temprana para el Modelo Más Simple:** Las curvas de aprendizaje del modelo con menor complejidad (`C = 1`) muestran una convergencia más rápida, lo que refleja una estabilización temprana en las tasas de error. Este comportamiento se debe a que el modelo simple es incapaz de capturar toda la variabilidad de los datos, lo que lleva a un aprendizaje menos detallado y, por ende, a una tasa de error que se estabiliza pronto.
- **Meseta de Error Más Alta para el Modelo Más Simple:** La meseta de error alcanzada por el modelo simple (`C = 1`) se mantiene a un nivel más alto que la de su contraparte más compleja (`C = 5`). Este patrón indica que el modelo simple tiene un sesgo elevado y no es lo suficientemente flexible para ajustarse bien a los datos, lo que se traduce en una tasa de error más alta. A pesar de converger rápidamente, el modelo no logra reducir su error significativamente.
- **Brecha entre Error de Entrenamiento y de Prueba:** La mayor brecha entre las tasas de error de entrenamiento y prueba en el modelo más complejo (`C = 5`) sugiere un sobreajuste. El modelo ha aprendido muy bien los detalles específicos de los datos de entrenamiento, pero no generaliza bien a datos no vistos. Por otro lado, el modelo simple (`C = 1`) tiene una brecha menor, lo que es característico de un modelo de alto sesgo que no puede ajustarse bien ni a los datos de entrenamiento.

Perspectivas Teóricas

- **Compensación de Sesgo-Varianza:** La relación entre el error de prueba y el error de entrenamiento es un reflejo directo de la **compensación sesgo-varianza**. Un modelo con bajo sesgo y alta varianza, como el de `C = 5`, tiende a sobreajustarse a los datos. En cambio, un modelo con alto sesgo y baja varianza, como el de `C = 1`, muestra un subajuste, ya que no puede capturar las complejidades de los datos. Encontrar un punto intermedio entre estas dos opciones, donde tanto el sesgo como la varianza sean bajos, es crucial para una buena generalización del modelo.
- **Complejidad del Modelo y Ajuste a los Datos:** Un modelo más complejo, como el que tiene `C = 5`, tiene mayor capacidad de aprender patrones complejos de los datos. Sin embargo, esto también lo hace más susceptible al sobreajuste, especialmente si la cantidad de datos de entrenamiento no es suficiente. El modelo más simple (`C = 1`) tiene menos capacidad para ajustar los datos, lo que genera un mayor sesgo, pero también es menos propenso a sobreajustarse. Esto refleja la clásica compensación entre un modelo que generaliza mal pero es robusto (alto sesgo) y uno que se ajusta perfectamente a los datos, pero con un rendimiento deficiente en datos nuevos (alta varianza).
- **Comprensión de las Curvas de Aprendizaje:** Las curvas de aprendizaje son fundamentales para diagnosticar el comportamiento de un modelo. Si las curvas continúan descendiendo, esto puede indicar que aún queda espacio para mejorar el modelo, posiblemente mediante la adición de más datos. Si las curvas se estabilizan en un punto por encima de la tasa de error mínima deseada, podría ser una señal de

que el modelo necesita mayor complejidad o que los datos no contienen suficiente información para una mejora significativa.

Conclusiones

La interpretación de estas curvas de aprendizaje proporciona valiosas lecciones sobre cómo ajustar un modelo y mejorar su rendimiento:

- Para el modelo más simple (`C = 1`), la tasa de error relativamente alta sugiere que podríamos necesitar incrementar la complejidad del modelo, por ejemplo, aumentando la profundidad del árbol o explorando nuevas características. Esta acción podría ayudar a reducir el sesgo y mejorar el rendimiento.
- Para el modelo más complejo (`C = 5`), la significativa diferencia entre los errores de entrenamiento y prueba indica que se debe considerar técnicas para prevenir el sobreajuste. Herramientas como la regularización, la poda de árboles o la obtención de más datos podrían ayudar a mejorar la capacidad de generalización del modelo. Además, el uso de validación cruzada o el ajuste de hiperparámetros puede ser útil para encontrar un balance adecuado.

En última instancia, el objetivo de un modelo de aprendizaje automático es lograr un balance óptimo entre sesgo y varianza. El sobreajuste y el subajuste son dos extremos que deben evitarse. Para lograr un modelo robusto, es esencial optimizar tanto la capacidad del modelo para capturar patrones como su habilidad para generalizar a nuevos datos.

Estas observaciones y ajustes no solo son aplicables a modelos de árboles de decisión, sino también a otros tipos de modelos en el ámbito del aprendizaje supervisado. La teoría de la compensación entre sesgo y varianza, así como la comprensión de las curvas de aprendizaje, son principios universales que guían la práctica del aprendizaje automático.

7.2 Sobreajuste

El sobreajuste (overfitting) es uno de los problemas más complejos y perjudiciales en el desarrollo de modelos de machine learning. Se presenta cuando un modelo se ajusta demasiado a los datos de entrenamiento, capturando no solo las tendencias subyacentes, sino también el ruido, las fluctuaciones aleatorias y las peculiaridades de esos datos. Esto sucede principalmente cuando un modelo es excesivamente complejo en relación con la cantidad de datos disponibles. Aunque un modelo sobreajustado muestra un rendimiento excelente en los datos de entrenamiento, su capacidad para generalizar a nuevos datos, es decir, a datos no vistos previamente, se ve gravemente afectada.

Causas del Sobreajuste El sobreajuste ocurre principalmente cuando la complejidad del modelo es demasiado alta en relación con el tamaño o la naturaleza de los datos. Existen varias razones por las cuales un modelo puede sobreajustarse:

1. **Exceso de parámetros**: Cuando un modelo tiene más parámetros de los necesarios para explicar los datos, puede ajustarse demasiado a los detalles finos de los datos de entrenamiento, incluidos los valores atípicos o el ruido.

2. **Modelo demasiado complejo**: Modelos con una alta capacidad de aprendizaje, como redes neuronales profundas o árboles de decisión muy profundos, pueden capturar patrones complejos, pero también pueden aprender detalles irrelevantes.
3. **Insuficiencia de datos**: Si el conjunto de datos de entrenamiento es pequeño, el modelo puede aprenderse demasiado bien los detalles específicos de esos pocos ejemplos y no generalizar adecuadamente a otros conjuntos de datos.
4. **Ruido en los datos**: Si los datos de entrenamiento contienen ruido (errores de medición, anomalías, valores atípicos), el modelo puede ajustarse a estos ruidos, reduciendo su capacidad para hacer predicciones correctas en datos nuevos.

Impacto del Sobreajuste en la Generalización La principal consecuencia del sobreajuste es la pérdida de capacidad de generalización. Un modelo generaliza bien cuando puede hacer predicciones precisas sobre datos que no ha visto antes. Si un modelo se ajusta demasiado a los datos de entrenamiento, puede perder esta capacidad de generalización, lo que se traduce en un aumento del error cuando se aplica a datos no vistos. La generalización es la habilidad del modelo para aplicar lo aprendido a nuevas situaciones o datos. Por ejemplo, un modelo de clasificación entrenado para identificar correos electrónicos spam puede funcionar perfectamente en los datos con los que se entrenó, pero si está sobreajustado, podría fallar al clasificar correctamente correos nuevos que no siguen exactamente los mismos patrones.

Curvas de Aprendizaje y su Relación con el Sobreajuste Las curvas de aprendizaje son herramientas gráficas útiles para analizar el comportamiento de un modelo de machine learning mientras cambia la complejidad del modelo o la cantidad de datos. A través de estas curvas, podemos observar cómo el error en los datos de entrenamiento y prueba evoluciona conforme aumentamos la complejidad del modelo. Las curvas de aprendizaje pueden ayudar a identificar si el modelo está sobreajustado o si está en el proceso de subajuste.

En el caso de **subajuste** (underfitting), el modelo es demasiado simple para capturar la complejidad de los datos. Esto se observa cuando tanto el error de entrenamiento como el de prueba son altos. A medida que aumentamos la complejidad del modelo, el error de entrenamiento disminuye, ya que el modelo es capaz de aprender los patrones presentes en los datos. Sin embargo, si la complejidad sigue aumentando más allá de un cierto punto, el modelo puede empezar a sobreajustarse, lo que se refleja en un aumento del error de prueba, a pesar de que el error de entrenamiento sigue disminuyendo.

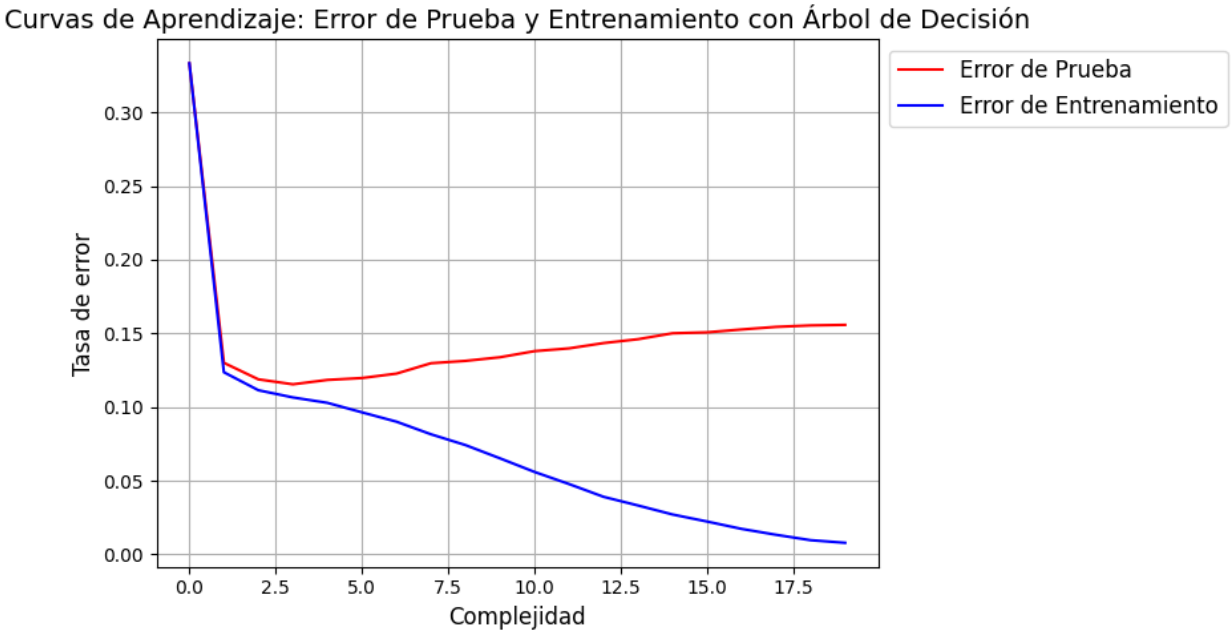
El Concepto de Complejidad del Modelo La **complejidad del modelo** se refiere a cuán sofisticado es un modelo en términos de su capacidad para aprender relaciones de los datos. Modelos más complejos, como los árboles de decisión profundos, redes neuronales con muchas capas o máquinas de soporte vectorial con parámetros ajustados, tienen mayor capacidad para capturar patrones complejos. Sin embargo, si esta capacidad no se controla adecuadamente, los modelos pueden aprender no solo las características verdaderas de los datos, sino también las irregularidades o el ruido que no se generaliza bien a datos no vistos.

Las curvas de aprendizaje permiten ilustrar visualmente cómo la complejidad afecta el rendimiento del modelo en los datos de prueba y entrenamiento. Idealmente, al aumentar la complejidad de un modelo, el error de prueba debería disminuir hasta llegar a un punto donde ya no haya mejoras o incluso comience a aumentar. Esto marca el inicio del sobreajuste, donde la capacidad del modelo para generalizar se ve afectada negativamente.

Identificación del Sobreajuste a través de Gráficas A continuación, analizaremos cómo se genera una curva de aprendizaje utilizando un modelo de árbol de decisión. Este modelo se entrenará con diferentes niveles de complejidad, y se graficará el error en los datos de entrenamiento y de prueba.

```
# Limpiar todas las variables del espacio de nombres actual para
iniciar con un espacio limpio.
%reset -f

# Importar las bibliotecas necesarias para operaciones numéricas y
graficación.
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics  # Para evaluar la precisión del modelo
from sklearn import tree  # Para usar modelos de árbol de decisión

# Establecer la semilla para la aleatorización (asegura resultados
repetibles).
np.random.seed(42)

# Definir la complejidad máxima, número de muestras para entrenamiento
y prueba, y número de iteraciones.
MAXC = 20  # Complejidad máxima (profundidad máxima del árbol)
N = 1000  # Número de muestras de entrenamiento
NTEST = 4000  # Número de muestras de prueba
ITERS = 3  # Número de iteraciones para promediar

# Inicializar matrices para almacenar las tasas de error de prueba y
entrenamiento.
yhat_test = np.zeros((ITERS, MAXC, 2))
yhat_train = np.zeros((ITERS, MAXC, 2))

# Repetir el experimento varias veces para obtener curvas de error
promedio más suaves.
for i in range(ITERS):
    # Generar datos sintéticos de entrenamiento a partir de una mezcla
de distribuciones normales.
    X_train = np.vstack([1.25 * np.random.randn(N, 2), 5 + 1.5 *
np.random.randn(N, 2), [8, 5] + 1.5 * np.random.randn(N, 2)])
    y_train = np.concatenate([np.ones((N, 1)), -np.ones((N, 1)),
np.ones((N, 1))])

    # Barajar aleatoriamente los datos de entrenamiento.
    perm = np.random.permutation(y_train.size)
    X_train = X_train[perm, :]
    y_train = y_train[perm]

    # Generar datos sintéticos de prueba usando las mismas
distribuciones que los datos de entrenamiento.
    X_test = np.vstack([1.25 * np.random.randn(NTEST, 2), 5 + 1.5 *
```

```
np.random.randn(NTEST, 2), [8, 5] + 1.5 * np.random.randn(NTEST, 2)])
    y_test = np.concatenate([np.ones((NTEST, 1)), -np.ones((NTEST,
1)), np.ones((NTEST, 1))])

    # Iterar sobre un rango de complejidades para el modelo de árbol
de decisión.
    for j, C in enumerate(range(1, MAXC + 1)):
        # Entrenar un clasificador de árbol de decisión con la
complejidad actual (profundidad).
        clf = tree.DecisionTreeClassifier(min_samples_leaf=1,
max_depth=C)
        clf.fit(X_train, y_train.ravel())  # Ajustar el modelo a los
datos de entrenamiento.

        # Evaluar el modelo en el conjunto de prueba y registrar la
tasa de error.
        yhat_test[i, j, 0] = 1. -
metrics.accuracy_score(clf.predict(X_test), y_test.ravel())
        # Evaluar el modelo en el conjunto de entrenamiento y
registrar la tasa de error.
        yhat_train[i, j, 0] = 1. -
metrics.accuracy_score(clf.predict(X_train), y_train.ravel())

# Graficar el error promedio de prueba (en rojo) y entrenamiento (en
azul) a lo largo de todas las iteraciones.
plt.figure(figsize=(9, 5))  # Establecer el tamaño de la figura para
una mejor visibilidad.

# Graficar los errores promedio
plt.plot(np.mean(yhat_test[:,:,0].T, axis=1), 'r', label='Error de
Prueba')
plt.plot(np.mean(yhat_train[:,:,0].T, axis=1), 'b', label='Error de
Entrenamiento')

# Etiquetas y leyenda
plt.xlabel('Complejidad', fontsize=12)
plt.ylabel('Tasa de error', fontsize=12)
plt.title('Curvas de Aprendizaje: Error de Prueba y Entrenamiento con
Árbol de Decisión', fontsize=14)
plt.legend(loc='upper left', fontsize=12, bbox_to_anchor=(1, 1))

# Habilitar la cuadrícula para una mejor legibilidad del gráfico.
plt.grid(True)

# Mostrar el gráfico.
plt.tight_layout()  # Ajustar el diseño para evitar solapamientos de
etiquetas.
plt.show()
```



Interpretación de las Curvas de Error

En el gráfico resultante, el **error de entrenamiento** (representado por la línea azul) disminuye continuamente a medida que aumenta la complejidad del modelo. Esto se debe a que un modelo más complejo tiene más capacidad para ajustar los datos, lo que le permite capturar patrones más precisos, incluyendo los ruidos o detalles no representativos. Sin embargo, esto no implica que un modelo con menor error de entrenamiento sea mejor. Un modelo que se ajusta demasiado bien a los datos de entrenamiento puede ser un indicio de que está sobreajustando, lo que puede resultar en una baja capacidad de generalización a nuevos datos. Es crucial que el modelo sea capaz de equilibrar el ajuste a los datos de entrenamiento sin sobreajustarse a ellos.

El **error de prueba** (representado por la línea roja) inicialmente disminuye conforme aumentamos la complejidad del modelo. Esto indica que el modelo está aprendiendo mejor los patrones generales de los datos. El error de prueba es una métrica crucial, ya que representa la capacidad del modelo para generalizar a datos no vistos durante el entrenamiento. Sin embargo, después de un cierto punto, el error de prueba comienza a aumentar, lo que indica que el modelo se ha ajustado demasiado a los detalles específicos de los datos de entrenamiento, es decir, ha sobreajustado. Este comportamiento es un indicador claro de que el modelo está aprendiendo patrones espúrios o detalles irrelevantes de los datos de entrenamiento que no tienen valor predictivo para los datos futuros.

Este fenómeno de aumento del error de prueba a medida que la complejidad del modelo sigue aumentando se conoce como el **sobreajuste**. Durante el sobreajuste, el modelo se ajusta excesivamente a los datos de entrenamiento, lo que lleva a que funcione muy bien en esos datos, pero no sea capaz de generalizar a datos nuevos, los cuales pueden diferir de las características específicas aprendidas del conjunto de entrenamiento. Es un fenómeno común cuando el modelo tiene demasiados parámetros en relación con la cantidad de datos de entrenamiento disponibles. En este caso, el modelo aprende detalles que no son representativos del comportamiento general de los datos, como el ruido o las fluctuaciones aleatorias.

Fenómeno del Sobreajuste

El **sobreajuste** se caracteriza por esta divergencia entre el error de entrenamiento y el error de prueba. Mientras que el modelo sigue mejorando en los datos de entrenamiento (con un error más bajo), su rendimiento en los datos de prueba empeora debido a que ha aprendido patrones espúrios o ruido que no se generalizan bien a otros conjuntos de datos. Este comportamiento es una indicación clara de que el modelo ha alcanzado un punto de complejidad excesiva. El modelo ya no está aprendiendo solo las características relevantes y generalizables de los datos, sino también los detalles específicos del conjunto de entrenamiento que no pueden replicarse en el mundo real.

A medida que el modelo sobreajusta, también es posible que el modelo empiece a perder su capacidad para identificar correctamente las características más importantes de los datos, ya que se enfoca en aprender características específicas, irrelevantes o no generalizables. Esto puede conducir a una baja precisión cuando el modelo se enfrenta a datos no vistos o a situaciones nuevas. El modelo no está bien preparado para adaptarse a las variaciones naturales de los datos del mundo real.

El sobreajuste también puede ocurrir debido a otros factores, como la presencia de **ruido en los datos**, la falta de datos suficientes, o el uso de algoritmos demasiado complejos sin mecanismos de regularización adecuados. Es importante tener en cuenta que el sobreajuste es un desafío inherente en el proceso de desarrollo de modelos predictivos, y abordarlo correctamente es crucial para la efectividad de los modelos en aplicaciones prácticas.

Conclusión

La clave para evitar el sobreajuste radica en encontrar el equilibrio adecuado entre la complejidad del modelo y su capacidad de generalización. La **complejidad óptima** del modelo es aquella que se encuentra justo antes de que el error de prueba comience a aumentar. En este punto, el modelo tiene suficiente capacidad para capturar las relaciones esenciales en los datos sin aprender el ruido innecesario. Sin embargo, el desafío está en identificar el punto exacto donde el modelo aún puede mejorar sin sacrificar su capacidad para generalizar.

Existen varias **técnicas de regularización** que ayudan a prevenir el sobreajuste, como el **poda de árboles de decisión**, que reduce la complejidad de los árboles de decisión limitando su profundidad o eliminando ramas que no aportan valor, o el uso de **regularización L1 y L2** en modelos lineales o redes neuronales. Estas técnicas penalizan la complejidad excesiva del modelo y ayudan a que el modelo mantenga solo las características más relevantes.

Otras estrategias para mitigar el sobreajuste incluyen el uso de **validación cruzada**, donde el conjunto de entrenamiento se divide en varios subconjuntos y el modelo se entrena y evalúa en diferentes combinaciones de estos subconjuntos. Esto permite evaluar cómo el modelo se comporta con diferentes particiones de los datos, lo que ayuda a identificar si el modelo tiene una tendencia a sobreajustarse a un subconjunto específico de los datos.

También se pueden emplear **ensembles** como **Random Forests** o **Boosting**, que combinan múltiples modelos más simples para mejorar la robustez y reducir el sobreajuste. Además, el uso de **dropout** en redes neuronales, donde se "apagan" aleatoriamente algunas neuronas durante el entrenamiento, ayuda a evitar que el modelo se ajuste demasiado a detalles específicos y ruidosos de los datos de entrenamiento.

La validación de un modelo no debe limitarse a evaluar su desempeño únicamente en los datos de prueba. En muchos casos, es recomendable usar **conjuntos de validación adicionales** o utilizar métodos de validación cruzada estratificada para obtener una evaluación más robusta de la capacidad de generalización del modelo.

En resumen, para evitar el sobreajuste, se debe considerar la **selección adecuada del modelo** y una **regularización efectiva**. Al mismo tiempo, las **curvas de aprendizaje** son herramientas valiosas para detectar señales de sobreajuste, ya que permiten monitorear cómo el rendimiento del modelo cambia con la complejidad. Un análisis cuidadoso de estas curvas ayuda a seleccionar el modelo adecuado, ajustando sus parámetros para obtener el mejor balance entre complejidad y capacidad de generalización.

8. Curas para el sobreajuste

El sobreajuste es un fenómeno que ocurre cuando el modelo se ajusta demasiado a los datos de entrenamiento, aprendiendo detalles y ruidos que no son representativos del problema en general. Es como memorizar las respuestas de un examen en lugar de comprender el material subyacente. Cuando el modelo se sobreajusta, obtiene muy buenos resultados en los datos con los que fue entrenado, pero su rendimiento en datos nuevos, que no ha visto antes, empeora considerablemente. Para prevenir esto, existen varias estrategias eficaces que ayudan a regular el ajuste del modelo y a mejorar su capacidad de generalización.

Estrategias Comunes para Prevenir el Sobreajuste:

- **Selección de Modelo mediante Ajuste de Hiperparámetros**: Los hiperparámetros son las configuraciones que determinan cómo funciona el modelo. Estos incluyen la profundidad de los árboles de decisión, el número de neuronas en una red neuronal o el valor de regularización en un modelo lineal. Ajustar estos parámetros es esencial para evitar que el modelo se vuelva demasiado complejo y, por ende, susceptible al sobreajuste.

  La selección de los hiperparámetros correctos es crucial. Un valor demasiado bajo puede llevar a un modelo demasiado simple (subajuste), mientras que un valor excesivo puede causar que el modelo se sobreajuste. Un enfoque eficaz para seleccionar los hiperparámetros adecuados es mediante **validación cruzada**. Esta técnica divide los datos en varios subconjuntos, entrena el modelo en algunos de ellos y valida su rendimiento en los otros. De esta forma, se evalúa la capacidad de generalización del modelo y se seleccionan los mejores hiperparámetros para maximizar su rendimiento en datos no vistos.

  Además, la **validación cruzada** permite comparar diferentes configuraciones de hiperparámetros para elegir la que mejor generalice a datos no observados. Al probar y ajustar continuamente, se obtiene un conjunto de parámetros que maximiza el rendimiento general del modelo sin sacrificar la capacidad de generalización.

- **Regularización**: La regularización es una técnica que agrega una penalización por complejidad durante el entrenamiento del modelo. Su propósito es evitar que el modelo se haga demasiado complejo, lo que puede llevar a que se sobreajuste a los detalles o ruidos del conjunto de entrenamiento.

  Piensa en la regularización como un conjunto de reglas que limitan la longitud de un ensayo. Al obligar a escribir de manera más concisa, se eliminan los detalles innecesarios y se enfocan solo en los puntos clave. De manera similar, la regularización penaliza los modelos por agregar complejidad innecesaria, favoreciendo aquellos modelos que son simples pero efectivos.

  Algunas técnicas comunes de regularización incluyen la **regularización L1** (Lasso) y **L2** (Ridge), que añaden términos de penalización basados en los coeficientes del modelo. Estas técnicas no solo ayudan a prevenir el sobreajuste, sino que también pueden mejorar la interpretabilidad del modelo al reducir el número de características importantes.

- **Conjuntos (Ensembles)**: Los conjuntos son una técnica que consiste en combinar las predicciones de varios modelos para mejorar el rendimiento general. En lugar de depender de un solo modelo, un enfoque de conjunto utiliza la "sabiduría de las multitudes" para hacer predicciones más robustas y precisas.

  Imagina que, en lugar de estudiar solo, formarías un grupo de estudio donde todos comparten sus puntos de vista para llegar a una conclusión más confiable. De manera similar, en un conjunto, se combinan las predicciones de varios modelos que pueden tener distintas fortalezas y debilidades. Dos enfoques comunes de conjuntos son el **bagging** y el **boosting**.

  - **Bagging** (Bootstrap Aggregating) implica entrenar múltiples modelos en subconjuntos aleatorios de los datos de entrenamiento, y luego combinar sus predicciones, generalmente por votación o promediado. Un ejemplo popular es el **Random Forest**, que utiliza múltiples árboles de decisión entrenados en distintas muestras de los datos.
  - **Boosting** funciona de manera diferente: cada modelo sucesivo intenta corregir los errores cometidos por el anterior. Los modelos se entrenan en secuencia, y los errores de los modelos previos reciben más peso en los siguientes entrenamientos. Métodos como **AdaBoost** y **Gradient Boosting** son ejemplos comunes de boosting.

Los conjuntos combinan los resultados de modelos simples para mejorar la precisión y reducir el riesgo de sobreajuste, ya que cada modelo individualmente puede ser propenso a sobreajustarse, pero al combinar sus predicciones, el riesgo general disminuye.

8.1 Cura I: Uso de la selección de modelos.

La **selección de modelos** es como elegir la herramienta adecuada para una tarea específica. Diferentes modelos tienen distintas formas de aprender y generalizar, y no todos son adecuados para todos los tipos de datos. Al seleccionar un modelo, uno de los factores clave es evaluar su

capacidad de generalización, medida por el **error de generalización** (E_{out}). El modelo que minimiza este error es el más adecuado para nuestra tarea.

> **Intuición:** El proceso de selección de los mejores hiperparámetros de un modelo es similar al de elegir el mejor clasificador. Imagina una caja de herramientas con diferentes configuraciones ajustables, donde cada herramienta tiene una gama de posibilidades (hiperparámetros) que pueden afectar su rendimiento. El objetivo es encontrar la combinación óptima de parámetros que mejoren la capacidad del modelo para aprender de los datos y generalizar a nuevos casos.

Un paso fundamental en este proceso es la **validación cruzada**, que permite evaluar cómo el modelo se comporta con diferentes particiones de los datos. Al dividir el conjunto de datos en varios subconjuntos y entrenar y evaluar el modelo en diferentes combinaciones, podemos asegurarnos de que el modelo no esté sobreajustado a un conjunto de datos específico y que pueda generalizar bien a nuevos datos.

El proceso de **validación de hiperparámetros** es esencial para afinar la capacidad de generalización del modelo. Ajustar los parámetros adecuadamente permite que el modelo se enfoque en aprender patrones generales sin perderse en los detalles específicos o en el ruido presente en los datos de entrenamiento.

A través de este enfoque, no solo estamos mejorando el rendimiento en los datos de entrenamiento, sino que estamos asegurando que nuestro modelo esté optimizado para prever resultados en nuevos escenarios. De este modo, utilizamos las técnicas de validación y selección de modelos para abordar el sobreajuste y mejorar la robustez general del modelo.

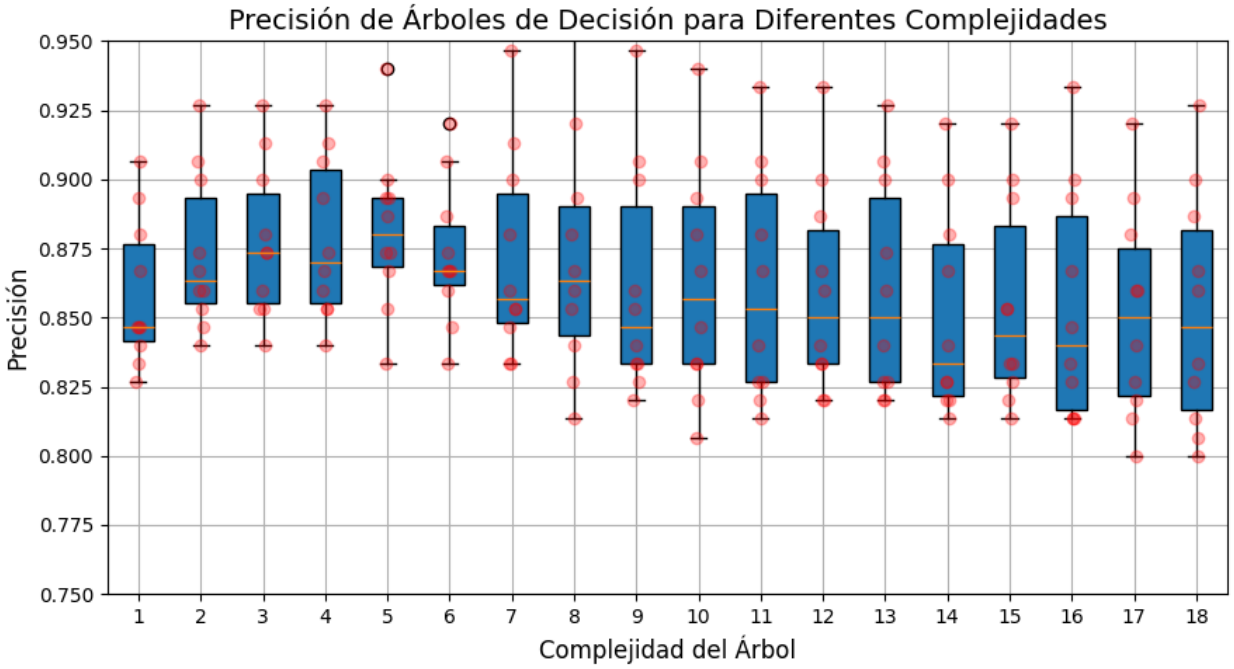
VIDEO: Validación cruzada

Al optimizar los hiperparámetros mediante la validación cruzada, encontramos un equilibrio entre la capacidad de aprendizaje del modelo y su habilidad para generalizar a nuevos datos. Esto nos ayuda a obtener un modelo que, aunque complejo, no se sobreajusta a los detalles innecesarios de los datos de entrenamiento.

```
# Limpiar todas las variables del espacio de nombres actual para
evitar conflictos.
%reset -f

# Importar los paquetes necesarios.
import numpy as np  # Para cálculos numéricos.
import matplotlib.pyplot as plt  # Para graficar.
from sklearn import metrics  # Para calcular métricas de rendimiento
como la precisión.
from sklearn import tree  # Para modelos de árbol de decisión.
from sklearn import model_selection  # Para utilidades de validación
cruzada y selección de modelos.

# Establecer la semilla para la aleatorización (asegura resultados
repetibles).
np.random.seed(42)

# Crear un conjunto de datos sintético para la demostración.
```

```
N = 500  # Número de muestras
# Generar puntos de distribuciones normales para formar nuestro
conjunto de datos.
X = np.vstack([1.25 * np.random.randn(N, 2), 5 + 1.5 *
np.random.randn(N, 2), [8, 5] + 1.5 * np.random.randn(N, 2)])
y = np.concatenate([np.ones((N, 1)), -np.ones((N, 1)), np.ones((N,
1))])

# Configurar una validación cruzada de 10 pliegues. Esto dividirá
nuestros datos en 10 partes.
kf = model_selection.KFold(n_splits=10, shuffle=True, random_state=0)

# Prepararse para buscar el mejor parámetro de complejidad dentro del
rango dado.
C = np.arange(2, 20)  # Rango de complejidades del modelo a evaluar.

# Inicializar una matriz para almacenar los puntajes de precisión para
diferentes pliegues y complejidades.
acc = np.zeros((10, len(C)))  # 10 para pliegues, 18 para el rango de
complejidades.

# Realizar validación cruzada.
for i, (train_index, val_index) in enumerate(kf.split(X)):
    # Dividir los datos en conjuntos de entrenamiento y validación
para el pliegue actual.
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    # Evaluar el modelo para cada nivel de complejidad.
    for j, c in enumerate(C):
        # Inicializar y entrenar un clasificador de árbol de decisión
en el nivel actual de complejidad.
        dt = tree.DecisionTreeClassifier(min_samples_leaf=1,
max_depth=c)
        dt.fit(X_train, y_train.ravel())  # Entrenar el modelo.
        yhat = dt.predict(X_val)  # Predecir los resultados del
conjunto de validación.

        # Calcular y almacenar la precisión para el pliegue y la
complejidad actual.
        acc[i, j] = metrics.accuracy_score(yhat, y_val)

# Graficar la precisión para cada complejidad como un diagrama de caja
para mostrar la distribución.
plt.figure(figsize=(9, 5))  # Establecer el tamaño de la figura para
una mejor visibilidad.
plt.boxplot(acc, vert=True, patch_artist=True, meanline=True)

# Superponer los puntajes de precisión individuales para cada nivel de
complejidad.
```

```
for i in range(len(C)):
    xderiv = (i + 1) * np.ones(acc[:, i].shape) + (np.random.rand(10,)
- 0.5) * 0.1
    plt.plot(xderiv, acc[:, i], 'ro', alpha=0.3)  # 'ro' indica
marcadores de círculo rojo.

# Establecer los límites para el eje y para centrarse en el rango de
interés.
plt.ylim(0.75, 0.95)

# Etiquetar los ejes y agregar título y leyenda.
plt.xlabel('Complejidad del Árbol', fontsize=12)
plt.ylabel('Precisión', fontsize=12)
plt.title('Precisión de Árboles de Decisión para Diferentes
Complejidades', fontsize=14)

# Mostrar el gráfico.
plt.grid(True)
plt.tight_layout()  # Ajustar el diseño para evitar solapamientos de
etiquetas.
plt.show()
```



En el aprendizaje automático, seleccionar el modelo adecuado es tan crucial como ajustar el modelo en sí. Es como elegir la llave correcta para una cerradura; la correcta encaja perfectamente y desbloquea el potencial del modelo. En nuestra búsqueda de este "ajuste perfecto", empleamos una técnica conocida como **validación cruzada**. Esta técnica es esencial porque nos ayuda a afinar los hiperparámetros del modelo, específicamente su complejidad, para asegurar que el modelo generalice bien a datos no vistos y evite el sobreajuste.

A continuación, te explico paso a paso lo que estamos haciendo en el código Python proporcionado:

1. **Creación de un Conjunto de Datos Sintético**: Comenzamos generando datos artificiales para controlar las características de los patrones subyacentes y el ruido. Este conjunto de datos sintético nos da un entorno controlado en el que podemos experimentar libremente con nuestros modelos. Esto es ventajoso porque elimina la incertidumbre que a veces acompaña al uso de datos reales, permitiéndonos enfocarnos exclusivamente en la calidad de los modelos y sus hiperparámetros.

2. **Configuración de la Validación Cruzada**: Utilizamos un método llamado **validación cruzada K-Fold**, que implica dividir nuestro conjunto de datos en 'K' partes (en este caso, 10). El modelo se entrena en 'K-1' partes y se valida en la parte restante, repitiendo este proceso hasta que cada subconjunto haya sido utilizado como conjunto de validación una vez. Esta técnica permite que cada dato tenga la oportunidad de ser utilizado tanto para entrenamiento como para validación, proporcionando una evaluación más robusta de la capacidad de generalización del modelo.

3. **Búsqueda de la Mejor Complejidad del Modelo**: Exploramos una gama de complejidades para nuestro modelo, que en este escenario es un clasificador de Árbol de Decisión. La complejidad de un árbol se controla ajustando su profundidad máxima, un parámetro que varía de 2 a 20 en este caso. Un árbol con baja complejidad (poca profundidad) podría no captar todos los patrones presentes en los datos, lo que conduciría al **subajuste**, mientras que un árbol con alta complejidad (gran profundidad) podría aprender demasiado sobre el ruido de los datos, lo que resultaría en **sobreajuste**.

4. **Entrenamiento y Validación del Modelo**: Para cada nivel de complejidad (profundidad del árbol), entrenamos un nuevo Árbol de Decisión y evaluamos su precisión en el conjunto de validación. Este proceso se repite para cada uno de los pliegues en nuestra configuración de validación cruzada, lo que nos permite obtener una estimación precisa de cómo se comporta el modelo con diferentes configuraciones.

5. **Análisis de los Resultados**: Después de entrenar el modelo en todos los pliegues y para cada nivel de complejidad, recopilamos las puntuaciones de precisión y las graficamos utilizando **diagramas de caja**. Estos diagramas nos muestran la distribución de las puntuaciones de precisión en cada nivel de complejidad, permitiéndonos observar cómo varía el rendimiento del modelo en diferentes pliegues. Además, agregamos puntos individuales al gráfico para visualizar las puntuaciones específicas de cada pliegue y analizar mejor la variabilidad.

6. **Elección de la Complejidad Óptima**: El objetivo es identificar el nivel de complejidad que logra un buen rendimiento consistente en todos los pliegues, es decir, un modelo que no solo se ajusta bien a los datos de entrenamiento, sino que también generaliza bien a nuevos datos. La complejidad óptima es aquella en la que

el modelo tiene un rendimiento equilibrado, sin caer en el sobreajuste ni el subajuste.

7. **Visualización del Resultado**: Finalmente, utilizamos un gráfico para visualizar los resultados obtenidos. El eje x muestra los diferentes niveles de complejidad del modelo (profundidad del árbol), mientras que el eje y representa la tasa de precisión. Los diagramas de caja nos permiten observar la variabilidad en el rendimiento a través de los pliegues, mientras que los puntos rojos indican las puntuaciones de precisión individuales para cada pliegue. Esta visualización facilita la identificación de la complejidad óptima que mejor balancea el aprendizaje de patrones y la capacidad de generalización.

A través de este proceso, el objetivo es lograr un modelo lo suficientemente complejo para aprender los patrones importantes pero lo suficientemente simple para ignorar el ruido y los detalles irrelevantes. Este equilibrio es crucial para crear un modelo que no solo se desempeñe bien en los datos de entrenamiento, sino que también sea capaz de generalizar eficazmente a nuevos datos, lo cual es la verdadera prueba de un buen modelo de aprendizaje automático.

```
# Calcular la precisión media en todos los pliegues de validación cruzada para cada nivel de complejidad.
mean_accuracy = np.mean(acc, axis=0)

# Encontrar el índice del nivel de complejidad que proporciona la mayor precisión media.
best_complexity_idx = np.argmax(mean_accuracy)

# Imprimir el nivel de complejidad y la precisión correspondiente.
# El índice se incrementa en 1 porque los niveles de complejidad comienzan desde 1, no desde 0.
print(f'La complejidad óptima es: {best_complexity_idx + 1} con una precisión media de: {mean_accuracy[best_complexity_idx]:.4f}')
```

```
La complejidad óptima es: 5 con una precisión media de: 0.8813
```

Este fragmento de código realiza el paso final en la selección del modelo mediante validación cruzada. Después de entrenar y validar el modelo a través de un rango de complejidades, determina qué nivel de complejidad dio la mejor precisión media, sugiriéndolo como la elección óptima para este problema en particular. La sentencia de impresión proporciona una salida clara y legible por humanos, mostrando la complejidad del modelo elegido y su precisión de validación, guiando los siguientes pasos en el flujo de trabajo del aprendizaje automático.

¿Cuál es el error de generalización esperado al seleccionar este método?

```
# Inicializar un clasificador de árbol de decisión con la complejidad óptima encontrada anteriormente.
# 'min_samples_leaf=1' garantiza que cada nodo hoja tendrá al menos una muestra.
# 'max_depth=idx+1' establece la profundidad del árbol según la mejor complejidad identificada en la validación cruzada.
```

```
dt = tree.DecisionTreeClassifier(min_samples_leaf=1,
max_depth=best_complexity_idx + 1)

# Ajustar el modelo de árbol de decisión a los datos de entrenamiento.
# El modelo aprende de los datos de entrenamiento usando la
complejidad óptima identificada.
dt.fit(X_train, y_train)

# Establecer el número de muestras para generar datos fuera de muestra
(nuevos, no vistos) y probar la generalización del modelo.
N_out_of_sample = 1000  # Número de muestras para el conjunto de
prueba fuera de muestra

# Generar datos fuera de muestra para evaluar el rendimiento del
modelo en datos no vistos.
# Estos datos siguen la misma distribución que los datos de
entrenamiento, pero son completamente nuevos para el modelo.
X_out_of_sample =
np.concatenate([1.25*np.random.randn(N_out_of_sample, 2),
5+1.5*np.random.randn(N_out_of_sample, 2)])
X_out_of_sample = np.concatenate([X_out_of_sample,
[8,5]+1.5*np.random.randn(N_out_of_sample, 2)])
y_out_of_sample = np.concatenate([np.ones((N_out_of_sample, 1)), -
np.ones((N_out_of_sample, 1))])
y_out_of_sample = np.concatenate([y_out_of_sample,
np.ones((N_out_of_sample, 1))])

# Usar el modelo entrenado para predecir las etiquetas de los datos
fuera de muestra.
y_pred = dt.predict(X_out_of_sample)

# Calcular e imprimir la precisión del modelo en los datos fuera de
muestra.
# Esto da una estimación de cuán bien el modelo generaliza a nuevos
datos (datos no vistos).
accuracy = metrics.accuracy_score(y_pred, y_out_of_sample)
print(f'Precisión en los datos fuera de muestra: {accuracy:.4f}')

Precisión en los datos fuera de muestra: 0.8707
```

Este fragmento de código muestra la evaluación final de un clasificador de árbol de decisión entrenado con la complejidad óptima determinada mediante validación cruzada. Al generar nuevos datos no vistos y comparar las predicciones del modelo con las etiquetas verdaderas, evaluamos la capacidad del modelo para generalizar más allá de los datos de entrenamiento. La sentencia print muestra la precisión del modelo en estos datos fuera de la muestra, proporcionando una medida cuantitativa de su rendimiento de generalización.

8.1.1 Train, Test y Validación

En el campo del aprendizaje automático, es fundamental comprender cómo utilizar los conjuntos de **train**, **test** y **validación** para desarrollar modelos robustos y evitar problemas como el sobreajuste. Cada uno de estos conjuntos tiene un papel específico en el proceso de entrenamiento y evaluación del modelo.

Datos de prueba (Test set)

El **conjunto de prueba** se utiliza **únicamente** para evaluar el rendimiento final del modelo. Después de entrenar y ajustar el modelo utilizando el conjunto de entrenamiento y validación, los datos de prueba se usan para obtener una **estimación imparcial** de cómo el modelo se comportará con datos nuevos. Los datos de prueba no deben usarse en ninguna fase del proceso de selección del modelo, ya que el uso de los mismos para ajuste o validación puede sesgar el rendimiento.

Ventajas:

- Permite obtener una estimación confiable del rendimiento real del modelo.
- Asegura que el modelo ha generalizado correctamente a nuevos datos.

Desventajas:

- No debe ser utilizado en la selección de modelos o la optimización de hiperparámetros.
- Si el conjunto de prueba es pequeño, puede no ser representativo de todos los datos posibles.

Datos de validación (Validation set)

El **conjunto de validación** se usa durante el proceso de entrenamiento para **ajustar los hiperparámetros** del modelo. Estos hiperparámetros incluyen la tasa de aprendizaje, el número de capas en redes neuronales, o parámetros específicos de un modelo como los de regularización. A diferencia de los datos de prueba, el conjunto de validación **interactúa directamente** con el proceso de entrenamiento, lo que permite optimizar el rendimiento del modelo antes de la evaluación final.

Ventajas:

- Permite la optimización del modelo sin comprometer el conjunto de prueba.
- Proporciona una **retroalimentación continua** sobre el rendimiento del modelo durante el entrenamiento.

Desventajas:

- Si se utiliza en exceso, puede llevar a la selección de un modelo que no generaliza bien, debido a la sobreajuste del conjunto de validación.
- No debe ser confundido con el conjunto de prueba, ya que su función es la optimización, no la evaluación final.

Datos de entrenamiento (Training set)

El **conjunto de entrenamiento** es donde el modelo realmente **aprende**. A partir de estos datos, el modelo ajusta sus parámetros internos (por ejemplo, los pesos en una red neuronal) para minimizar una función de **pérdida** o **error**. A través de este proceso, el modelo identifica patrones en los datos que le permiten realizar predicciones. Sin embargo, el riesgo es que si el modelo es demasiado complejo, puede **sobajustarse** a los detalles de los datos de entrenamiento, perdiendo su capacidad de generalización.

Ventajas:

- Permite al modelo aprender patrones complejos y representaciones útiles.
- Es esencial para entrenar cualquier modelo de aprendizaje automático.

Desventajas:

- Si se tiene un modelo muy complejo o un conjunto de datos limitado, puede llevar a un **sobreajuste**.
- Un modelo entrenado solo en este conjunto puede ser muy bueno en esos datos pero fallar al aplicarse en nuevos datos.

8.2 Cura II: Uso de la Regularización

La **regularización** es una técnica crucial para mejorar la capacidad de generalización del modelo, evitando el sobreajuste. Existen varios tipos de regularización, siendo los más conocidos **L1 (Lasso)** y **L2 (Ridge)**.

Regularización L2 (Ridge)

La **regularización L2**, también conocida como **Ridge**, penaliza los coeficientes grandes de los modelos, añadiendo un término proporcional al **cuadrado** de la magnitud de los coeficientes a la función de pérdida. Este enfoque tiende a **reducir** la magnitud de los coeficientes, pero **no los hace cero**. Así, las características no son eliminadas, sino que su peso se reduce, lo que lleva a modelos más simples y menos susceptibles al sobreajuste.

Ventajas:

- No elimina características, lo que puede ser útil cuando se cree que todas las características tienen algún nivel de relevancia.
- Funciona bien cuando hay muchas características y se sospecha que algunas pueden tener un impacto pequeño.

Desventajas:

- No realiza **selección de características**, por lo que todas las características se mantienen en el modelo, lo que puede resultar en modelos más complejos.
- Si las características irrelevantes no se reducen adecuadamente, pueden introducir ruido.

Regularización L1 (Lasso)

La **regularización L1**, o **Lasso**, penaliza la **magnitud absoluta** de los coeficientes. A diferencia de L2, L1 puede **hacer que algunos coeficientes sean exactamente cero**, eliminando efectivamente

esas características del modelo. Esta propiedad es útil para la **selección automática de características** y puede ayudar a obtener modelos más simples y menos susceptibles al sobreajuste.

Ventajas:

- Realiza **selección de características** automáticamente, eliminando las irrelevantes.
- Ideal para situaciones con **alta dimensionalidad**, donde muchas características no son relevantes.

Desventajas:

- Puede eliminar características relevantes si la penalización es demasiado fuerte.
- En algunos casos, puede ser menos estable que L2, especialmente si hay muchas características correlacionadas.

Comparación entre L1 y L2

- **L2** tiende a funcionar mejor cuando todas las características son relevantes, ya que solo reduce sus magnitudes, pero no las elimina.
- **L1** es más útil cuando se cree que solo algunas características son relevantes, ya que elimina las irrelevantes por completo.

8.3 Cura III: Métodos de Conjunto (Ensemble)

Los métodos de **conjunto** combinan las predicciones de múltiples modelos para mejorar la **precisión** y **robustez** del sistema. Estos métodos se dividen principalmente en **bagging** y **boosting**, cada uno con sus propias características y ventajas.

Bagging (Bootstrap Aggregating)

Bagging es una técnica que busca reducir la **varianza** de un modelo. Se entrena múltiples modelos en **subconjuntos diferentes** del conjunto de entrenamiento, generados mediante **muestreo con reemplazo** (bootstrap). Luego, las predicciones de estos modelos se combinan, generalmente mediante **promedio** (para regresión) o **votación mayoritaria** (para clasificación). El bagging es muy efectivo cuando el modelo base tiene alta varianza, como los **árboles de decisión**.

Un ejemplo común de bagging es el **Bosque Aleatorio (Random Forest)**, que entrena múltiples árboles de decisión y luego combina sus predicciones para obtener un resultado final más robusto.

Ventajas:

- Reduce la **varianza** y mejora la precisión del modelo.
- Se puede aplicar a modelos con alta varianza (por ejemplo, árboles de decisión).
- Ayuda a mejorar el rendimiento de los modelos en problemas complejos.

Desventajas:

- Requiere mayor **potencia computacional** debido al entrenamiento de múltiples modelos.
- Puede ser menos interpretativo, ya que combina muchos modelos independientes.

Boosting

Boosting es una técnica que construye modelos **secuencialmente**, donde cada nuevo modelo intenta corregir los errores cometidos por los modelos anteriores. En cada iteración, se da mayor peso a las instancias que fueron mal clasificadas por los modelos previos, lo que permite **mejorar el rendimiento en datos difíciles**. Los métodos más conocidos de boosting son **AdaBoost** y **Gradient Boosting**.

Boosting convierte modelos **débiles** (modelos que tienen un rendimiento ligeramente mejor que el azar) en un modelo **fuerte**, al aprender secuencialmente de los errores de los modelos anteriores. Algunos ejemplos populares son **XGBoost** y **LightGBM**.

Ventajas:

- Reduce tanto el **sesgo** como la **varianza** del modelo.
- A menudo produce **modelos de alto rendimiento** en comparación con métodos individuales.
- Es capaz de manejar datos complejos y desbalanceados.

Desventajas:

- Es susceptible al **sobreajuste** si no se controla adecuadamente la complejidad del modelo.
- Puede ser más **costoso computacionalmente** que otros métodos debido a la naturaleza secuencial del entrenamiento.
- Puede ser difícil de interpretar debido a su complejidad.

Comparación entre Bagging y Boosting

- **Bagging** es ideal cuando se quiere reducir la varianza, especialmente con modelos de alta varianza como los árboles de decisión. Su enfoque es **paralelizable**, ya que los modelos se entrenan de forma independiente.
- **Boosting** se utiliza para **reducir el sesgo**, enfocándose en las instancias difíciles de predecir y ajustando secuencialmente el modelo. Tiene el potencial de mejorar significativamente el rendimiento, pero puede ser más susceptible al sobreajuste y requiere más tiempo computacional.

La comprensión y correcta aplicación de estos métodos puede tener un gran impacto en el rendimiento de un modelo de aprendizaje automático. Desde la división adecuada de los datos en entrenamiento, validación y prueba, hasta la elección de técnicas de regularización y métodos de conjunto, cada estrategia tiene su papel en la construcción de modelos robustos que generalicen bien a nuevos datos. Sin embargo, es importante tener en cuenta las ventajas y desventajas de cada enfoque y aplicarlos en función de las características específicas del problema y los datos disponibles.

9. ¿Qué hacer cuando...?

Cuando nos enfrentamos a problemas con nuestro modelo, como un **alto sesgo** o **alta varianza**, es importante contar con estrategias claras para mejorar el rendimiento del clasificador. A continuación se presentan algunas recomendaciones detalladas para abordar estos problemas.

... nuestro algoritmo muestra un alto **sesgo**.

Un alto **sesgo** generalmente significa que el modelo no está **aprendiendo** correctamente, ya que está asumiendo supuestos demasiado fuertes sobre los datos. Esto a menudo ocurre cuando se usa un modelo **demasiado simple** para el problema en cuestión, lo que impide que el algoritmo capture la complejidad de los datos.

Posibles soluciones:

- **Agregar más características**: Si podemos generar nuevas características que capturen mejor las variaciones importantes de los datos, esto podría ayudar a reducir el sesgo. Las **características discriminantes** podrían proporcionar la información adicional necesaria para que el modelo sea más flexible y se ajuste mejor a los datos.
  - Ejemplo: En un problema de clasificación de imágenes, agregar características como color, textura o formas puede hacer que el modelo sea más capaz de discriminar entre las clases.
- **Usar un modelo más sofisticado**: Un alto sesgo puede ser señal de que estamos utilizando un modelo demasiado simple para nuestros datos. En este caso, se puede optar por modelos más complejos que tengan mayor capacidad para aprender patrones más complejos en los datos.
  - Ejemplo: Si estamos usando un modelo lineal (como la regresión lineal o la regresión logística), podríamos probar con modelos no lineales, como **máquinas de soporte vectorial (SVM)** o **redes neuronales**.
  - También podemos explorar modelos más complejos que capturan mejor las relaciones entre las variables, como los **árboles de decisión** o los **modelos de bosque aleatorio (Random Forest)**.
- **Reducir la regularización**: Si estamos usando técnicas de regularización, como **L1 (Lasso)** o **L2 (Ridge)**, y nuestro modelo presenta un alto sesgo, podría ser necesario disminuir la penalización sobre los parámetros del modelo. Esto permitiría que el modelo se ajuste mejor a los datos, aunque con el riesgo de aumentar la varianza.
  - Ejemplo: Disminuir el valor del hiperparámetro de regularización en un modelo de regresión lineal.
- **Usar menos muestras**: En algunos casos, entrenar el modelo con **menos muestras** puede ayudar a mejorar el tiempo de entrenamiento y reducir el sesgo. Sin embargo, esto no garantiza una mejora significativa del rendimiento, ya que la capacidad del modelo para ajustarse a los datos sigue siendo limitada.
  - Esta técnica es útil principalmente cuando se trata de modelos que no necesitan una gran cantidad de datos para generalizar, pero su efectividad puede ser limitada.

... nuestro algoritmo muestra **alta varianza**.

Una alta **varianza** significa que el modelo está **sobajustándose** a los datos de entrenamiento, aprendiendo demasiado sobre el ruido en los mismos y perdiendo capacidad de generalización a nuevos datos.

Posibles soluciones:

- **Usar menos características**: Reducir la cantidad de características o realizar una **selección de características** puede ayudar a disminuir el sobreajuste. Al eliminar las

características irrelevantes o redundantes, el modelo se vuelve menos propenso a aprender patrones erróneos.
    - Ejemplo: Usar métodos de **reducción de dimensionalidad** como **Análisis de Componentes Principales (PCA)** o técnicas de **selección de características** para eliminar características que no contribuyen significativamente al modelo.
- **Usar un modelo más simple**: Si el modelo es demasiado complejo, como un modelo de red neuronal con demasiadas capas o un árbol de decisión profundo, puede ser útil cambiar a un modelo más simple. Modelos más simples tienen menos capacidad de sobreajustarse y, por lo general, generalizan mejor.
    - Ejemplo: Usar un modelo de **regresión logística** en lugar de un **árbol de decisión** si el conjunto de datos es relativamente pequeño y no muy complejo.
    - También se pueden ajustar los hiperparámetros para disminuir la complejidad del modelo, como el número de capas o unidades en redes neuronales.
- **Usar más muestras de entrenamiento**: Aumentar el tamaño del conjunto de datos de entrenamiento proporciona más ejemplos para que el modelo aprenda. Esto puede ayudar a reducir la varianza, ya que al tener más datos, el modelo tiene más información para generalizar y menos probabilidades de ajustarse al ruido.
    - Ejemplo: Si el conjunto de datos es pequeño, se puede aumentar mediante **aumento de datos (data augmentation)** en tareas de visión por computadora o utilizando técnicas de **muestreo**.
- **Usar técnicas de conjunto (Ensemble methods)**: Las técnicas de conjunto como el **bagging** y el **boosting** ayudan a reducir la varianza al combinar múltiples modelos. Estos métodos permiten que un grupo de modelos, con errores diferentes, se unan para producir una predicción más robusta.
    - **Bagging**: Utiliza técnicas como el **Bootstrap Aggregating**, donde se entrenan varios modelos en subconjuntos aleatorios de los datos de entrenamiento, y luego sus predicciones se combinan.
        - Ejemplo: **Random Forests** es un ejemplo típico de bagging.
    - **Boosting**: Enfoque secuencial donde cada nuevo modelo trata de corregir los errores de los modelos anteriores, lo que ayuda a mejorar el rendimiento y reduce la varianza de la predicción.
        - Ejemplo: **Gradient Boosting** o **XGBoost**.

Técnicas de Regularización:

- Si la **varianza** es alta, puede ser útil aplicar técnicas de regularización para reducir la complejidad del modelo y forzarlo a ser más general.
    - **Regularización L2 (Ridge)**: Disminuye los coeficientes de las características sin llevarlos a cero, lo que reduce la complejidad del modelo.
    - **Regularización L1 (Lasso)**: Puede eliminar características irrelevantes al reducir sus coeficientes a cero, lo que hace que el modelo sea más simple y menos propenso a sobreajuste.

En resumen:

Cuando enfrentamos problemas de **sesgo alto**, podemos mejorar la capacidad de aprendizaje del modelo al agregar más características, usar modelos más complejos o reducir la regularización. Mientras que en el caso de **alta varianza**, podemos simplificar el modelo, usar

técnicas de selección de características, aumentar los datos de entrenamiento o aplicar métodos de conjunto para mejorar la robustez del modelo. La clave está en encontrar el balance adecuado para que el modelo pueda generalizar bien a datos nuevos sin sobreajustarse ni subajustarse.