



Ajustes de Modelos y Validación Cruzada

En el mundo del aprendizaje automático, lograr que los modelos funcionen bien no solo con datos de entrenamiento sino también con información nueva es fundamental. Esta capacidad de "generalización" distingue a un modelo útil de uno engañosamente perfecto.

A lo largo de esta presentación, exploraremos los tipos de error en el aprendizaje automático, el dilema entre sesgo y varianza, y las técnicas de validación cruzada que nos permiten evaluar correctamente el rendimiento de nuestros modelos.

R por Kibernum Capacitación S.A.

Preguntas de Activación

Influencia del Preprocesamiento

En la sesión anterior revisamos el preprocesamiento de datos, como la codificación de variables y el escalado. ¿Cómo crees que estos pasos podrían influir en la forma en que un modelo se ajusta o se sobreajusta?

Recomendaciones "Perfectas"

¿Has tenido la sensación de que una aplicación o recomendación es "demasiado perfecta", como si supiera exactamente lo que quieres? ¿Podría eso relacionarse con el sobreajuste?

Limitaciones de la Evaluación

¿Por qué crees que no es suficiente entrenar un modelo y evaluar su precisión solo con los datos usados en el entrenamiento?

Tipos de Error en el Aprendizaje Automático

Sobreajuste (Overfitting)

El modelo memoriza los datos de entrenamiento en lugar de aprender patrones generales. Como un estudiante que se aprende las respuestas de memoria pero no entiende los conceptos.

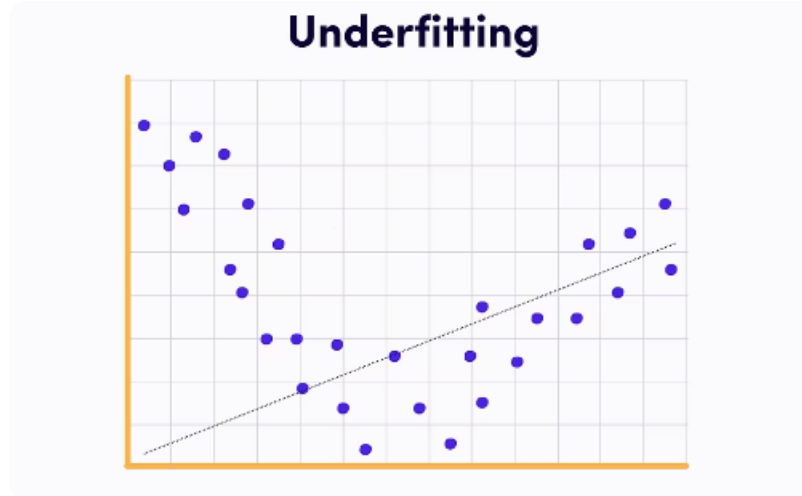
Consecuencias: Excelente rendimiento en entrenamiento pero mal rendimiento con datos nuevos. No generaliza, solo repite lo que ya vio.

Subajuste (Underfitting)

El modelo es demasiado simple y no captura la complejidad del problema. Como un estudiante que solo lee los títulos sin profundizar en el contenido.

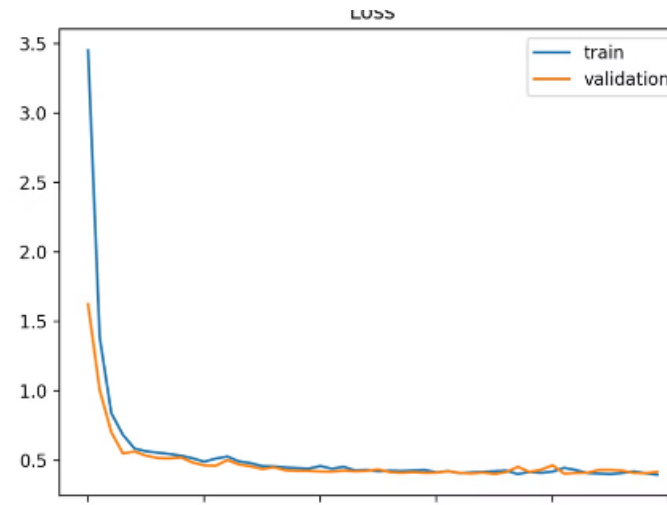
Consecuencias: Mal desempeño tanto en entrenamiento como en prueba. Ignora relaciones clave entre variables.

Comparación Visual: Ajuste de Modelos



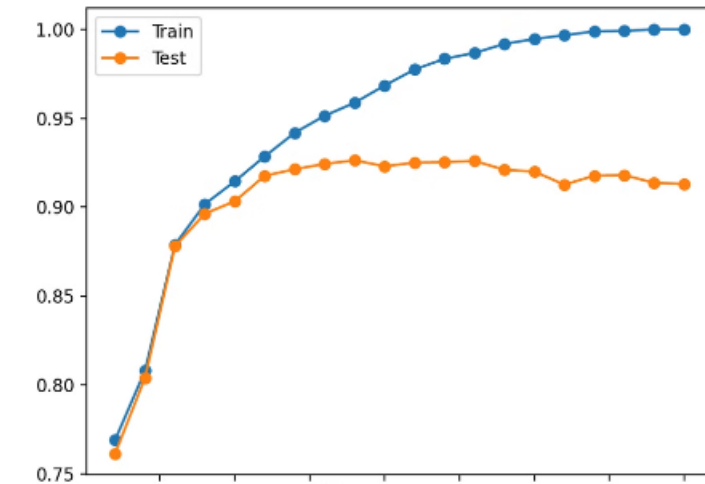
Subajuste

El modelo es demasiado simple y no logra capturar la relación entre los datos. La línea recta no puede representar adecuadamente la distribución de los puntos, resultando en un error alto tanto en entrenamiento como en prueba.



Ajuste Adecuado

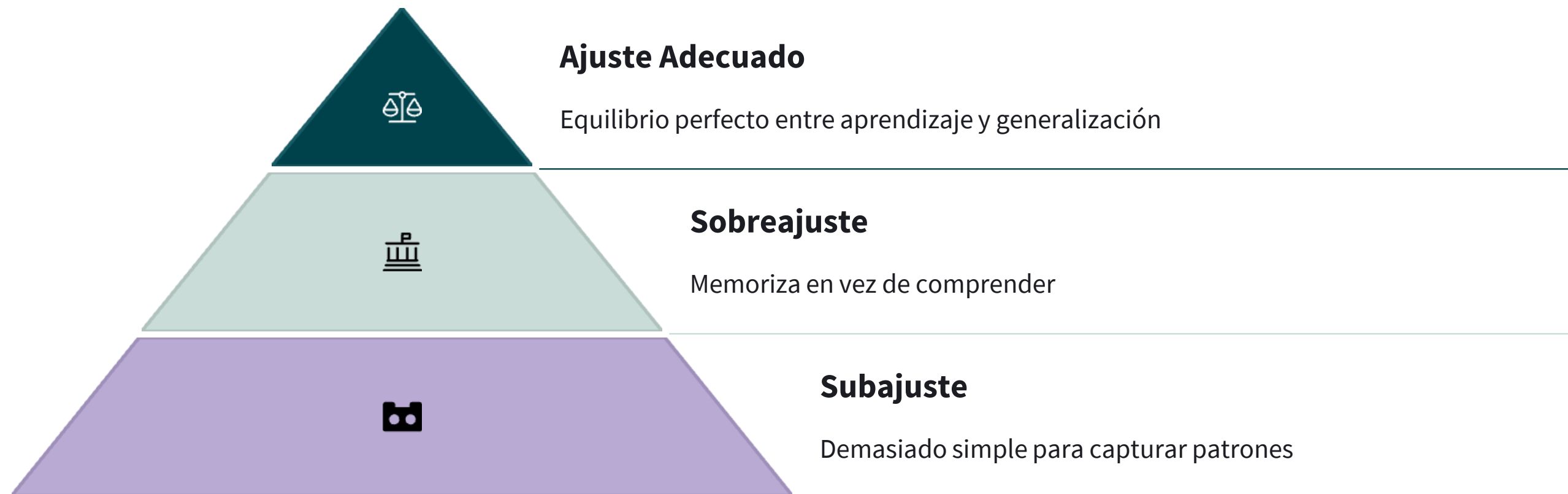
El modelo generaliza correctamente, identificando patrones sin exagerar. Captura la tendencia general de los datos sin dejarse influenciar por el ruido, logrando un buen equilibrio entre precisión y capacidad de generalización.



Sobreajuste

El modelo memoriza demasiado los datos de entrenamiento, lo que perjudica su rendimiento con nuevos datos. La curva pasa por todos los puntos pero tiene una forma exagerada y poco natural.

¿Qué Significa Ajustar un Modelo?



Ajustar un modelo significa entrenarlo para que reconozca patrones en los datos. El desafío está en encontrar el punto óptimo: si aprende muy poco, no captará los patrones importantes (subajuste); si aprende demasiado, puede terminar "memorizando" los datos (sobreajuste).

Un buen modelo no memoriza, comprende. Ni el que "no sabe nada" (subajuste), ni el que "se lo sabe todo de memoria" (sobreajuste) son útiles. El verdadero aprendizaje está en el equilibrio.



Sobreajuste: Cuando el Modelo Memoriza



Definición

Un modelo sobreajustado es como un estudiante que se aprendió de memoria todas las respuestas del ensayo, pero no entendió realmente los temas.



Causas

Ocurre cuando el modelo es demasiado complejo para la cantidad o tipo de datos disponibles. En vez de aprender patrones importantes, también aprende el "ruido".



Consecuencias

Excelente rendimiento en entrenamiento pero mal rendimiento en datos nuevos. Las predicciones se vuelven erráticas, impredecibles y poco confiables.

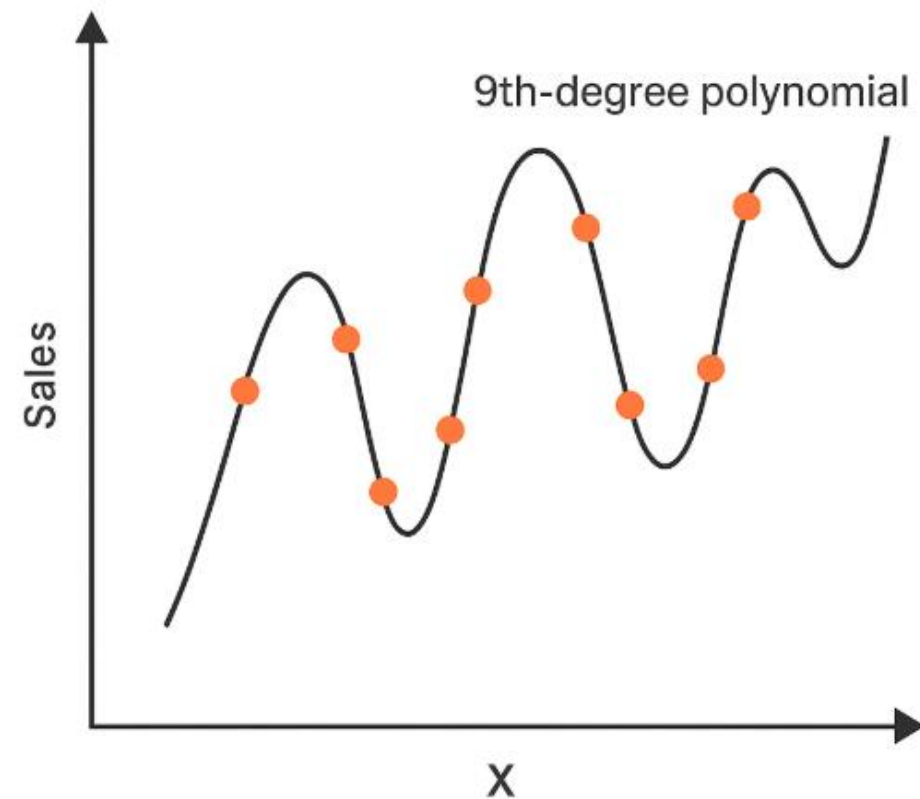
Sobreajuste: Cuando el Modelo Memoriza

Imagina que tienes 10 datos sobre ventas y quieres predecir una tendencia.

Un modelo equilibrado trazaría una línea o curva que resume bien la tendencia general.


Pero si haces que el modelo pase por cada uno de esos 10 puntos exactos, crearás una curva exagerada que no sigue ningún patrón lógico, solo conecta los puntos por obligación.

Resultado: el modelo parece muy preciso, pero si llega un nuevo dato, no sabe qué hacer. Las predicciones se vuelven erráticas, impredecibles y poco confiables.



Sobreajuste

La imagen muestra cómo un modelo polinomial extremadamente complejo intenta ajustarse exactamente a cada punto de entrenamiento. Aunque la curva pasa por todos los datos, su forma es exagerada y poco natural. Este tipo de modelo tiende a fallar al enfrentarse a nuevos datos, ya que no ha aprendido la tendencia general, sino que ha memorizado el ruido. Es un ejemplo clásico de sobreajuste: alta precisión en entrenamiento, pero baja capacidad de generalización.



Subajuste: Cuando el Modelo es Demasiado Simple

Definición

Un modelo subajustado es como un estudiante que leyó solo los títulos de los temas, sin profundizar. Cuando le hacen preguntas, responde de forma vaga o demasiado general.

Causas

Sucede cuando el modelo es demasiado simple para la complejidad del problema. No tiene la capacidad suficiente para reconocer ni siquiera los patrones básicos en los datos.

Consecuencias

Mal desempeño tanto en entrenamiento como en prueba. Ignora relaciones clave entre las variables, resultando en predicciones pobres sin importar qué datos se utilicen.

Ajuste Adecuado: El Punto Ideal

Este es el escenario ideal. El modelo ha aprendido lo suficiente para capturar los patrones verdaderos del problema, pero no tanto como para memorizar los detalles irrelevantes.

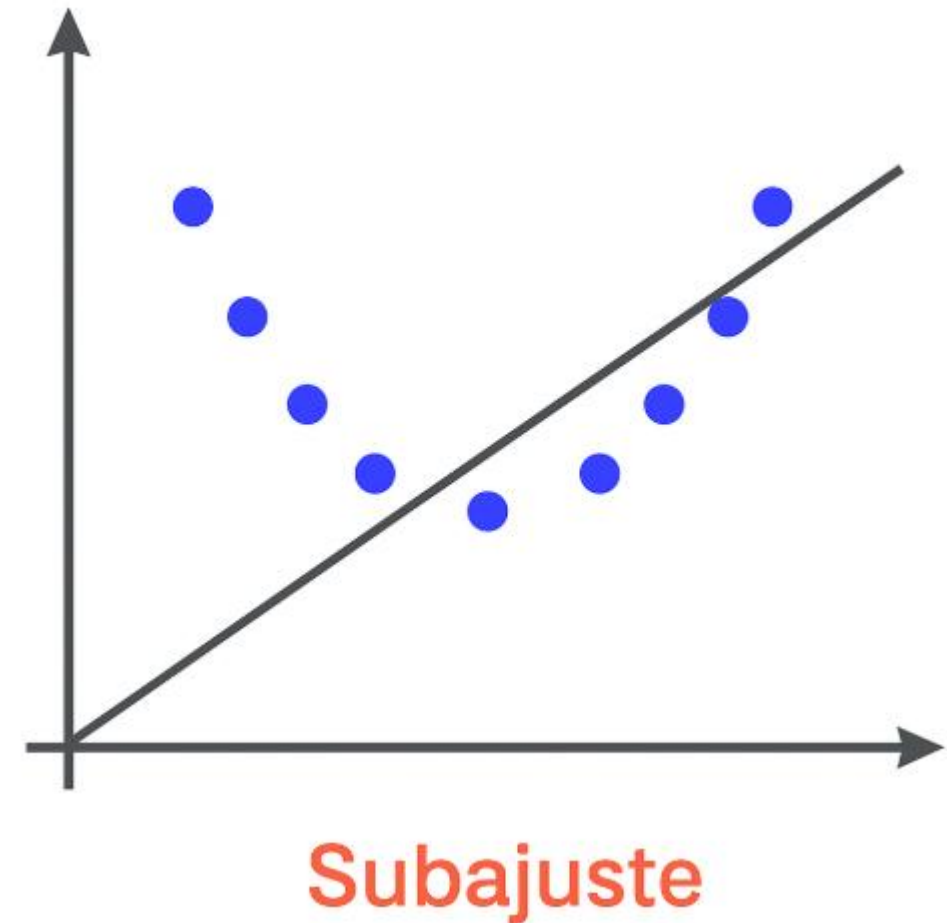
¿Cómo se logra?

- Eligiendo un modelo apropiado para la tarea (ni muy simple ni muy complejo).
- Usando técnicas como validación cruzada para medir su rendimiento de forma equilibrada.
- Reservando un conjunto de prueba independiente.
- Ajustando hiperparámetros para afinar su comportamiento.
- Tener un modelo que rinda bien tanto en entrenamiento como en datos nuevos, logrando así una buena capacidad de generalización.

Un buen modelo no memoriza, comprende.

Ni el que “no sabe nada” (subajuste), ni el que “se lo sabe todo de memoria” (sobreajuste) son útiles.

El verdadero aprendizaje está en el equilibrio.



Ejercicio Guiado: Explorando el Ajuste del Modelo en Función de su Complejidad

Objetivo del ejercicio

Comprender cómo la complejidad de un modelo afecta su capacidad para aprender de los datos y generalizar correctamente. A través de visualizaciones comparativas, podrán identificar los efectos del subajuste, ajuste adecuado y sobreajuste, utilizando modelos de regresión polinomial con distintos grados.

Contexto del ejercicio

Vamos a trabajar con un conjunto de datos sintético generado con `make_regression`, el cual incluye ruido (variabilidad aleatoria) para simular un entorno más realista.

Luego, construiremos modelos de regresión polinomial de distintos grados para visualizar cómo cambia su comportamiento al aumentar la complejidad.

Ejercicio Guiado: Explorando el Ajuste del Modelo en Función de su Complejidad

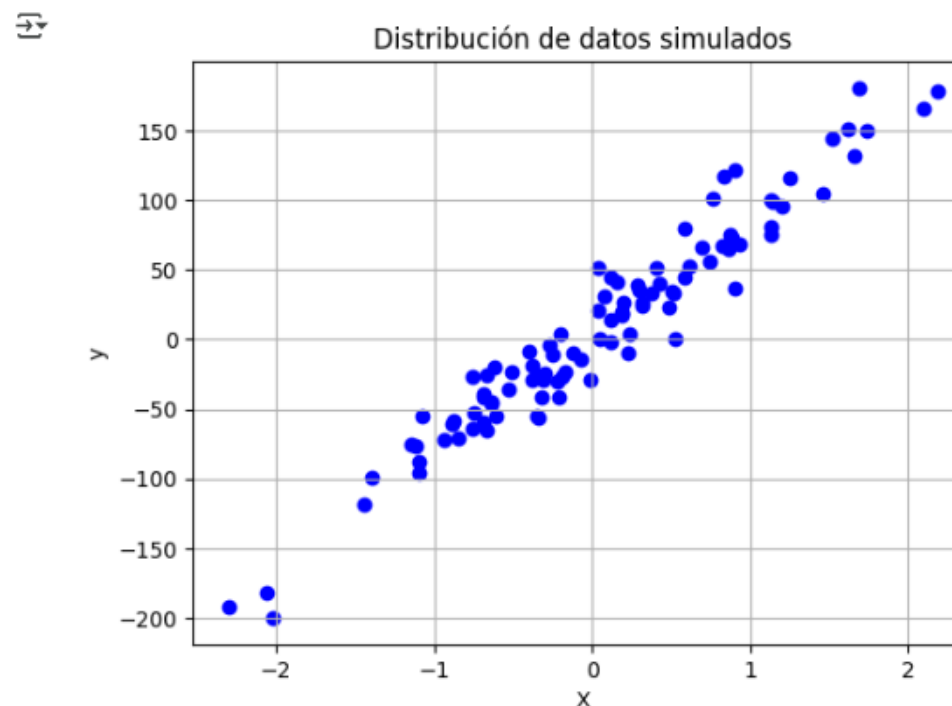
Paso 1: Generar y visualizar los datos

Los puntos generados simulan una relación lineal con algo de ruido. Esta visualización inicial será la base para observar cómo distintos modelos intentan ajustarse a estos datos.

```
# Paso 1: Importamos las librerías necesarias
from sklearn.datasets import make_regression
import matplotlib.pyplot as plt
import numpy as np

# Generamos un conjunto de datos sintético
# • 100 muestras
# • 1 característica (X)
# • Ruido aleatorio para simular datos reales
X, y = make_regression(n_samples=100, n_features=1, noise=20, random_state=1)

# Visualizamos los datos en un diagrama de dispersión
plt.scatter(X, y, color='blue')
plt.title("Distribución de datos simulados")
plt.xlabel("X")
plt.ylabel("y")
plt.grid(True)
plt.show()
```



Ejercicio Guiado: Explorando el Ajuste del Modelo en Función de su Complejidad

Paso 2: Entrenar modelos con distintos grados de complejidad

```
[2] # Paso 2: Entrenamos modelos de regresión polinomial con diferentes grados

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Definimos tres niveles de complejidad: simple (1), medio (4), complejo (15)
grados = [1, 4, 15]

# Creamos una figura con 3 gráficos lado a lado
plt.figure(figsize=(12, 4))

# Para cada grado, entrenamos y visualizamos el modelo correspondiente
for i, grado in enumerate(grados):
    # Convertimos X en variables polinomiales según el grado
    poly = PolynomialFeatures(degree=grado)
    X_poly = poly.fit_transform(X)

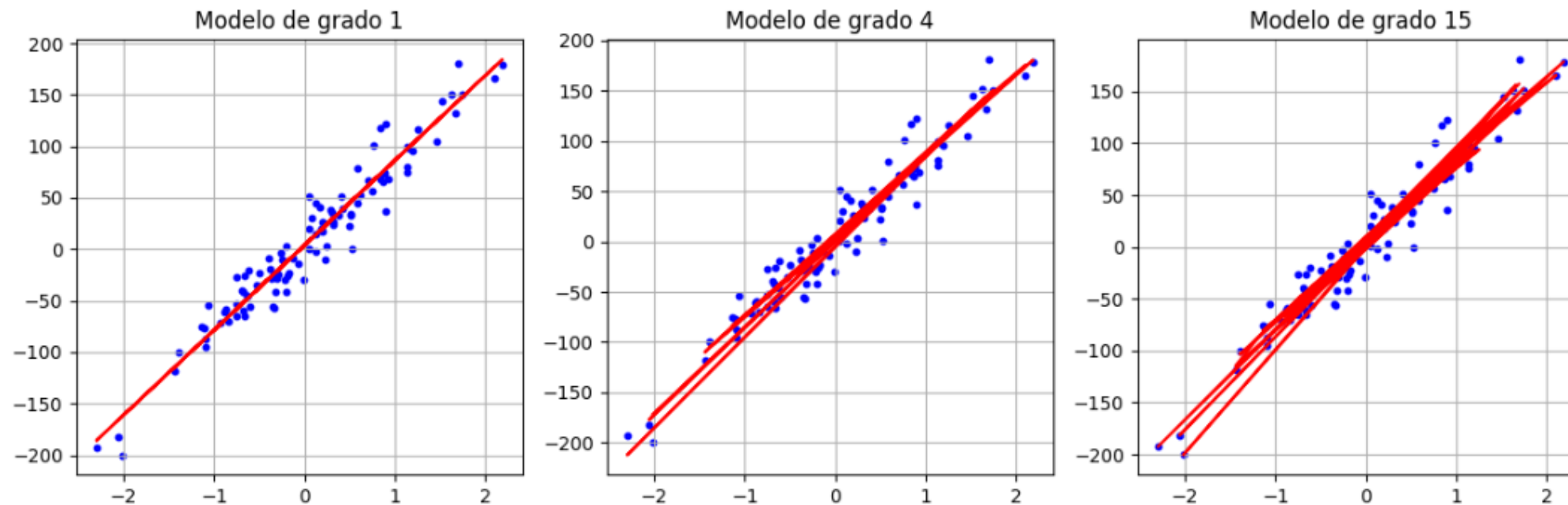
    # Creamos y entrenamos el modelo
    modelo = LinearRegression().fit(X_poly, y)

    # Realizamos predicciones con los mismos datos
    y_pred = modelo.predict(X_poly)

    # Dibujamos el gráfico: puntos reales (azul) y predicciones (línea roja)
    plt.subplot(1, 3, i+1)
    plt.scatter(X, y, color='blue', s=10)
    plt.plot(X, y_pred, color='red')
    plt.title(f'Modelo de grado {grado}')
    plt.grid(True)
    plt.tight_layout()

# Mostramos los 3 gráficos juntos
plt.show()
```


Ejercicio Guiado: Explorando el Ajuste del Modelo en Función de su Complejidad



Comparamos tres modelos con distinta capacidad de aprendizaje:

Grado 1: Modelo lineal muy simple.

Grado 4: Modelo más flexible, capaz de adaptarse a la forma general.

Grado 15: Modelo muy complejo, capaz de memorizar incluso el ruido.

Ejercicio Guiado: Explorando el Ajuste del Modelo en Función de su Complejidad

Observemos las curvas generadas y respondamos en conjunto:

- ¿Cuál de los modelos parece subajustado (no sigue la forma de los datos)?
- ¿Cuál muestra un ajuste adecuado (captura la tendencia general sin exagerar)?
- ¿Cuál parece sobreajustado (se adapta demasiado incluso al ruido)?

Ahora podemos concluir:

- Grado 1: modelo demasiado simple → subajuste.
- Grado 4: buen equilibrio → ajuste adecuado.
- Grado 15: modelo demasiado complejo → sobreajuste.

Sesgo y Varianza: El Gran Dilema

Sesgo (Bias)

Se refiere a los errores que comete un modelo por hacer suposiciones simplificadas sobre los datos. Un modelo con alto sesgo no logra representar la complejidad real del problema.

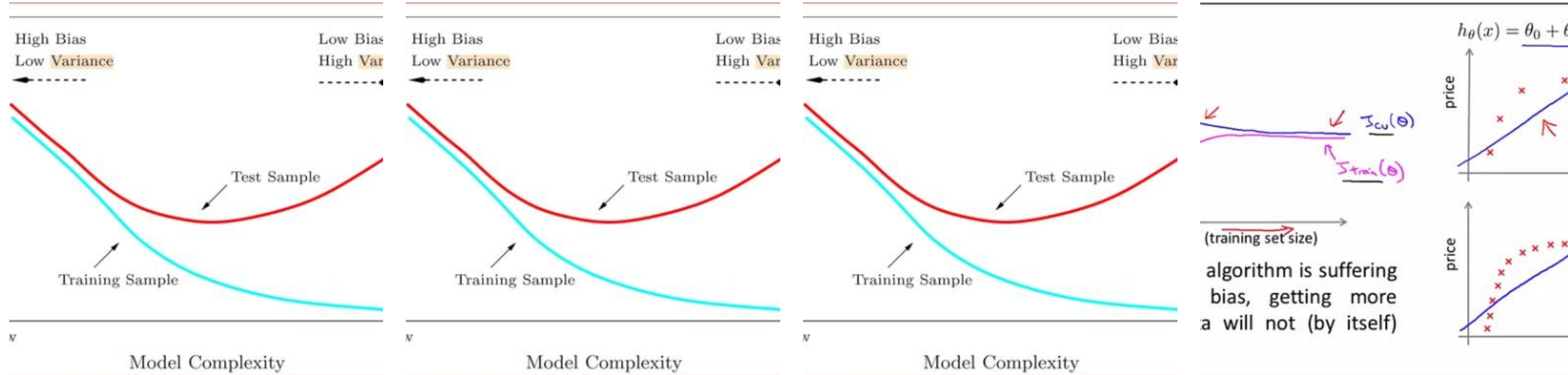
Características: Aprende poco o nada, es rígido y no se adapta a los datos, tiende al subajuste.

Varianza

Mide la sensibilidad del modelo a pequeñas variaciones en los datos de entrenamiento. Un modelo con alta varianza cambia mucho si se entrena con distintos subconjuntos de datos.

Características: Aprende demasiado, es muy complejo y se adapta incluso al ruido, tiende al sobreajuste.

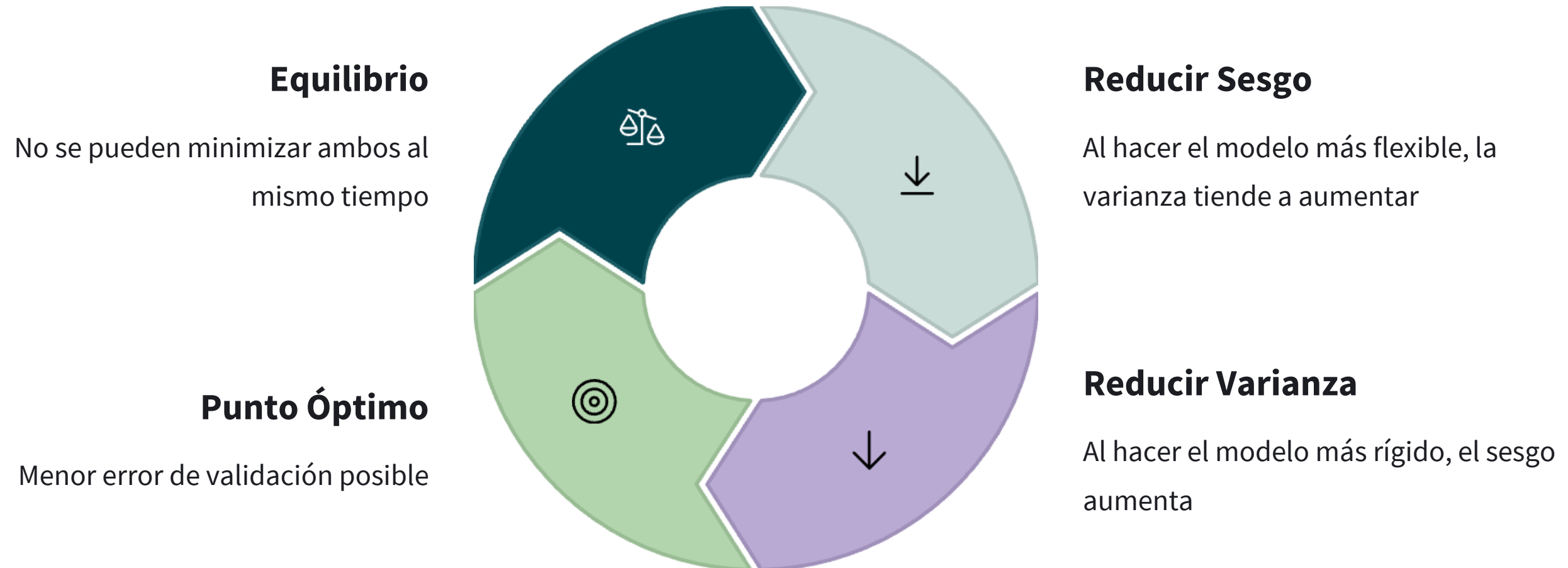
Visualización del Dilema Sesgo-Varianza



Esta visualización utiliza una metáfora de tiro al blanco para representar cómo se comportan diferentes modelos en términos de sesgo y varianza:

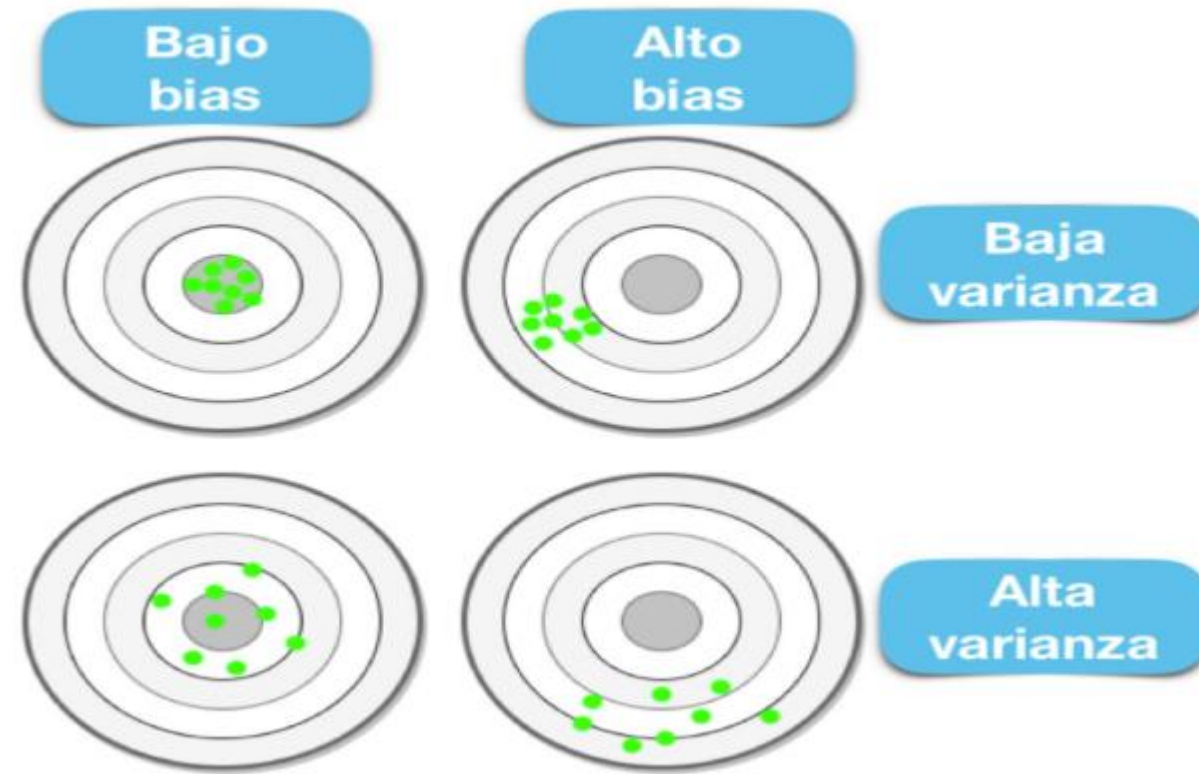
Bajo sesgo, baja varianza: predicciones precisas y consistentes (modelo ideal). Alto sesgo, baja varianza: predicciones consistentes pero lejos del objetivo (subajuste). Bajo sesgo, alta varianza: modelo inestable con predicciones variables (sobreajuste). Alto sesgo, alta varianza: predicciones malas e inestables (peor escenario).

Trade-Off Entre Sesgo y Varianza



El objetivo del aprendizaje automático es encontrar el punto óptimo donde el modelo tenga el menor error de validación posible, logrando un equilibrio entre sesgo y varianza. Este balance es lo que convierte al Machine Learning en un arte técnico.

Trade-Off Entre Sesgo y Varianza



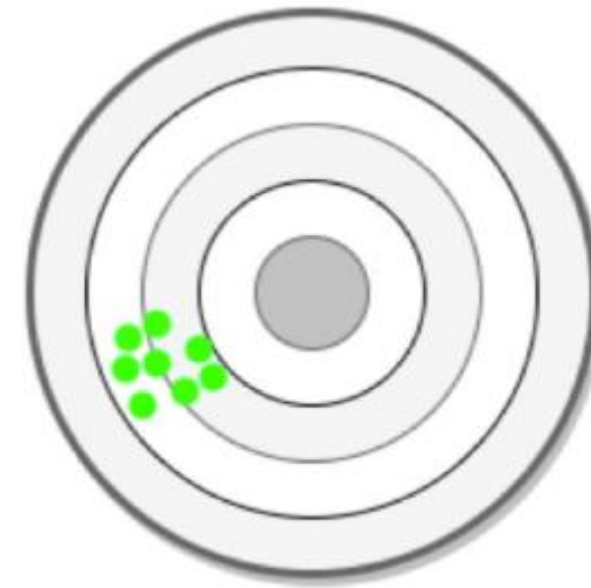
Esta imagen, tomada del sitio [Acturio \(2022\)](#), representa cómo se comportan diferentes modelos de Machine Learning en términos de bias (sesgo) y varianza utilizando una metáfora de tiro al blanco:

Trade-Off Entre Sesgo y Varianza

Bajo bias, baja varianza: los disparos (predicciones) son precisos y consistentes: el modelo aprende y generaliza bien.

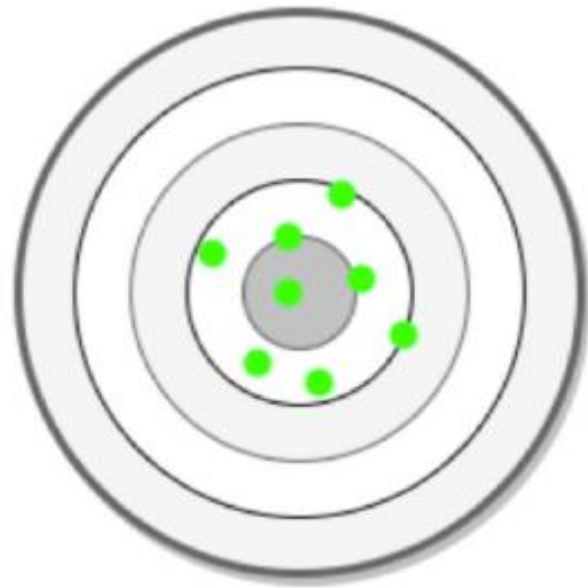


Alto bias, baja varianza: las predicciones son consistentes, pero están lejos del objetivo: el modelo no capta la complejidad del problema (subajuste).

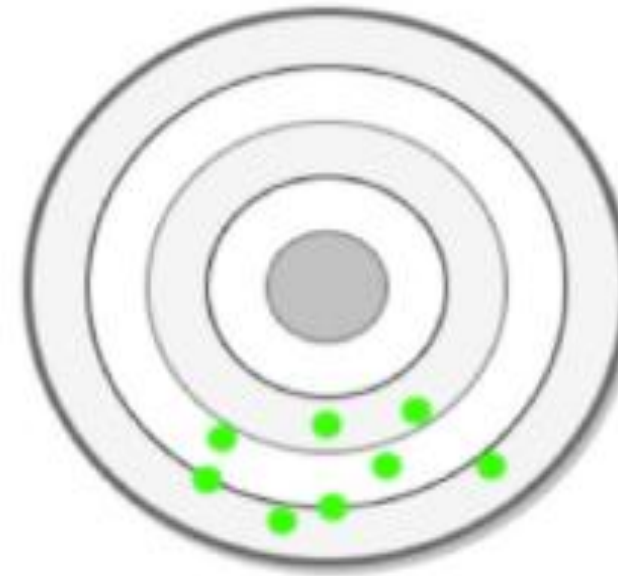


Trade-Off Entre Sesgo y Varianza

Bajo bias, alta varianza: el modelo aprende, pero es inestable: las predicciones varían demasiado con cambios en los datos (sobreajuste).



Alto bias, alta varianza: predicciones malas e inestables: el peor de los escenarios.



Ejercicio Práctico: Explorando Sesgo y Varianza en Modelos Polinomiales

Este ejercicio tiene como objetivo ayudarte a visualizar el comportamiento del sesgo y la varianza al variar la complejidad de un modelo de regresión polinomial.

A través de un ejemplo práctico, comprenderás cómo un modelo demasiado simple puede subajustarse (alto sesgo) y cómo uno demasiado complejo puede sobreajustarse (alta varianza).

Paso 1: Importación de librerías y generación de datos

```
{x} 0s [0] # Paso 1: Importar las librerías necesarias para el ejercicio
      from sklearn.datasets import make_regression          # Generar datos sintéticos para regresión
      from sklearn.preprocessing import PolynomialFeatures  # Para generar variables polinomiales
      from sklearn.linear_model import LinearRegression    # El modelo de regresión lineal
      from sklearn.pipeline import make_pipeline          # Permite combinar transformación + modelo
      from sklearn.model_selection import cross_val_score  # Para aplicar validación cruzada
      import matplotlib.pyplot as plt                     # Para graficar los resultados
      import numpy as np                                  # Para trabajar con arreglos numéricos

      # Creamos un conjunto de datos con 1 sola variable (X) y una respuesta (y)
      # Añadimos ruido (noise=20) para simular condiciones del mundo real
      X, y = make_regression(n_samples=100, n_features=1, noise=20, random_state=0)
```

Generamos un dataset simple con 1 variable explicativa (X) y un objetivo (y), agregando ruido para simular condiciones reales. Esto nos permitirá ver cómo distintos modelos se comportan frente a datos con variabilidad.

Ejercicio Práctico: Explorando Sesgo y Varianza en Modelos Polinomiales

Paso 2: Entrenar modelos con diferentes grados y evaluar su desempeño

```
✓ [3] # Paso 2: Evaluar modelos de regresión polinomial con distintos grados de complejidad
0s

grados = range(1, 15)    # Probaremos modelos desde grado 1 (línea recta) hasta grado 14 (muy complejos)
val_scores = []          # Lista para guardar los errores de validación cruzada

# Recorremos todos los grados y entrenamos un modelo para cada uno
for grado in grados:
    # Creamos un pipeline que:
    # 1. Transforma los datos con PolynomialFeatures
    # 2. Aplica un modelo de regresión lineal sobre esos datos transformados
    modelo = make_pipeline(PolynomialFeatures(degree=grado), LinearRegression())

    # Evaluamos el modelo usando validación cruzada (5 particiones del dataset)
    # Esto nos da una idea de qué tan bien generaliza el modelo
    scores = cross_val_score(modelo, X, y, cv=5, scoring='neg_mean_squared_error')

    # Calculamos el promedio de los errores obtenidos y lo almacenamos
    # (negamos el resultado porque sklearn devuelve el error negativo)
    val_scores.append(-scores.mean())
```

Entrenamos múltiples modelos de regresión polinomial, comenzando con los más simples (grado 1) y avanzando hacia modelos mucho más complejos (hasta grado 14).

Para cada modelo, evaluamos su rendimiento utilizando validación cruzada, una técnica que nos permite medir qué tan bien generaliza el modelo a nuevos datos.

Esto es clave para entender el equilibrio entre sesgo y varianza: qué tanto aprende el modelo y qué tan estable es cuando cambia el conjunto de entrenamiento.

Ejercicio Práctico: Explorando Sesgo y Varianza en Modelos Polinomiales

Paso 3: Visualizar el trade-off entre sesgo y varianza

```
[4] # Paso 3: Visualizar el error según la complejidad del modelo

# Graficamos los errores de validación en función del grado del polinomio
plt.plot(grados, val_scores, label='Error de Validación (CV)', color='red')

# Etiquetas y título del gráfico
plt.xlabel("Grado del polinomio")           # Complejidad del modelo
plt.ylabel("Error cuadrático medio")         # Medida del error
plt.title("Trade-off entre sesgo y varianza") # Interpretación del gráfico

# Estética del gráfico
plt.legend()
plt.grid()
plt.show()
```

El gráfico muestra cómo varía el error de validación a medida que aumenta la complejidad del modelo:

- 1. Al principio, el modelo es demasiado simple → alto sesgo, alto error.*
- 2. Luego, el modelo mejora → ajuste adecuado, menor error.*
- 3. Finalmente, el modelo es demasiado complejo → alta varianza, el error vuelve a aumentar por sobreajuste.*

Observen que existe un punto óptimo donde el error de validación es más bajo. Ese punto representa el equilibrio entre sesgo y varianza, y es ahí donde el modelo generaliza mejor.



Estrategias para Reducir el Trade-Off

Estrategia	Efecto
Aumentar la cantidad de datos	Reduce la varianza
Seleccionar un modelo más simple	Reduce la varianza, pero puede aumentar el sesgo
Aplicar regularización (L1, L2)	Controla la complejidad del modelo
Validación cruzada	Detecta sobreajuste antes de poner en producción

El trade-off entre sesgo y varianza no es un obstáculo, sino una guía. Comprenderlo nos permite ajustar el modelo justo donde es más útil: ni tan simple que ignore los datos, ni tan complejo que los memorice.

No buscamos el modelo más complejo ni el más preciso en entrenamiento, sino el más confiable en producción.



Técnicas de Validación Cruzada



¿Por qué validar un modelo?

Porque lo importante no es que el modelo recuerde lo que ya vio, sino que sepa cómo actuar frente a nuevos casos.



Objetivo principal

Estimar de manera confiable el rendimiento del modelo y evitar tanto el sobreajuste como el subajuste.



Técnicas principales

Retención simple (Holdout), Validación Cruzada K-Fold, Validación cruzada aleatoria (Shuffle Split) y Leave-One-Out (LOOCV).

Retención Simple (Holdout)



Conjunto de datos completo

Todos los datos disponibles



División de datos

Separación en entrenamiento y prueba



Proporciones típicas

80% entrenamiento, 20% prueba

La retención simple o holdout es la forma más básica de validación. Se divide el conjunto de datos en dos partes: una para entrenar el modelo y otra para evaluarlo. Es rápida y fácil de implementar, pero inestable ya que el resultado depende mucho de cómo se dividieron los datos.

Esta técnica puede desperdiciar información si el dataset es pequeño, ya que una parte significativa de los datos no se utiliza para el entrenamiento.

Retención Simple (Holdout)

```
# Paso 1: Importar las librerías necesarias
from sklearn.datasets import make_regression          # Para crear datos simulados
from sklearn.model_selection import train_test_split # Para dividir el dataset
import pandas as pd                                  # Para mostrar los datos en formato de tabla

# Paso 2: Generar datos sintéticos para regresión
# Creamos 100 muestras con 1 sola característica y algo de ruido (simula datos reales)
X, y = make_regression(n_samples=100, n_features=1, noise=15, random_state=1)

# Paso 3: Dividir los datos en conjuntos de entrenamiento (80%) y prueba (20%)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Paso 4: Imprimir el tamaño de cada subconjunto
print("Tamaño de los conjuntos:")
print(f"X_train: {X_train.shape}, y_train: {y_train.shape}")
print(f"X_test: {X_test.shape}, y_test: {y_test.shape}")

# Paso 5: Mostrar una muestra de los datos
print("\n Primeras 5 filas del conjunto de entrenamiento:")
df_train = pd.DataFrame({'X': X_train.flatten(), 'y': y_train})
print(df_train.head())

print("\n Primeras 5 filas del conjunto de prueba:")
df_test = pd.DataFrame({'X': X_test.flatten(), 'y': y_test})
print(df_test.head())
```

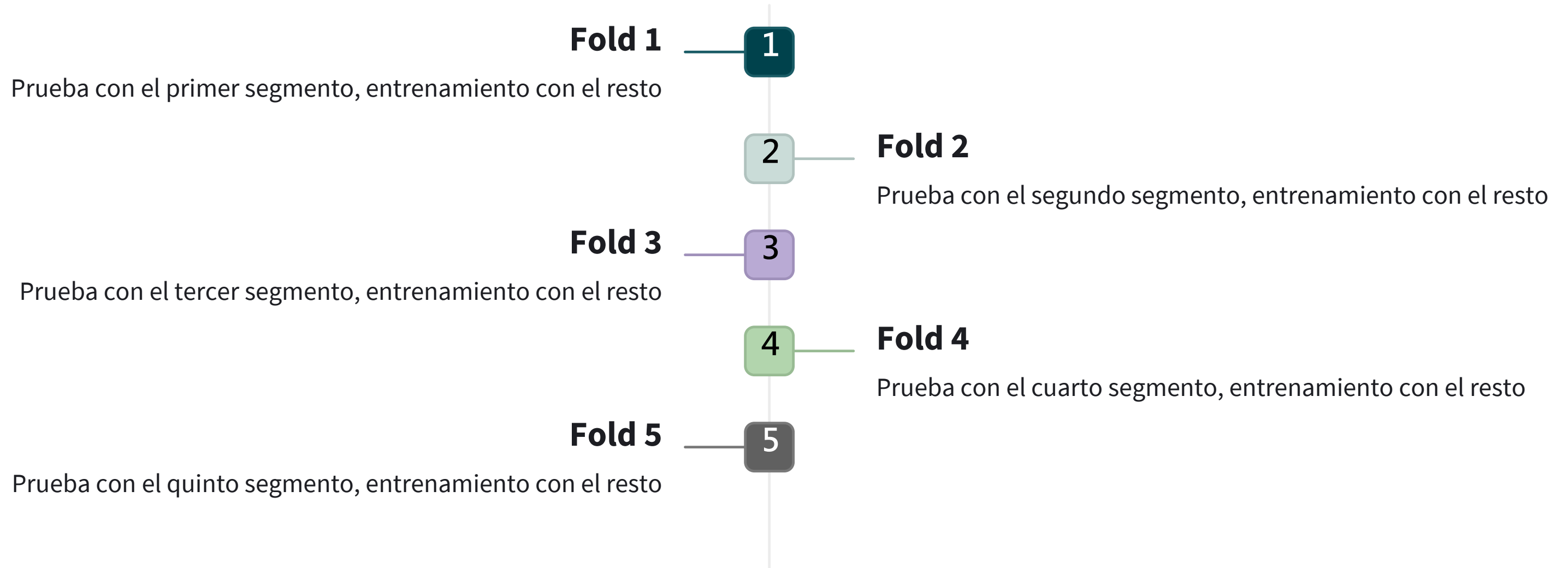
Ventajas:

- *Rápido y fácil de implementar.*

Desventajas:

- *Inestable: el resultado depende mucho de cómo se dividieron los datos.*
- *Puede desperdiciar información si el dataset es pequeño.*

Validación Cruzada K-Fold



La validación cruzada K-Fold es una de las técnicas más utilizadas y más robustas que el holdout. Divide los datos en K partes iguales (llamadas folds) y entrena el modelo K veces, usando K-1 folds como entrenamiento y 1 como prueba, calculando luego el promedio de los resultados.

Sus ventajas incluyen usar todos los datos tanto para entrenar como para validar, proporcionando resultados más estables y representativos.

Esquema Visual del Proceso de K-Fold Cross Validation

```
✓ [8] # Importamos las funciones necesarias de scikit-learn
0s from sklearn.model_selection import cross_val_score, KFold # cross_val_score para evaluar el modelo con validación cruzada
from sklearn.linear_model import LinearRegression # Usaremos regresión lineal como modelo de ejemplo

# Creamos una instancia del modelo de regresión lineal
modelo = LinearRegression()

# Definimos el esquema de validación cruzada K-Fold
# n_splits=5 → dividimos el dataset en 5 partes (folds)
# shuffle=True → mezclamos los datos antes de dividirlos (importante para que no queden ordenados)
# random_state=1 → asegura que los resultados sean reproducibles (la misma división cada vez)
kf = KFold(n_splits=5, shuffle=True, random_state=1)

# Aplicamos validación cruzada al modelo con el esquema definido
# cv=kf → usamos el objeto KFold como esquema de validación
# scoring='r2' → usamos el coeficiente de determinación R² como métrica (0 a 1, donde 1 es mejor)
scores = cross_val_score(modelo, X, y, cv=kf, scoring='r2')

# Mostramos los resultados obtenidos en cada uno de los 5 folds
print(f"Resultados por fold: {scores}")

# Calculamos y mostramos el promedio de R² como una medida general de desempeño
print(f"Promedio R²: {scores.mean():.2f}")
```

➡ Resultados por fold: [0.96139046 0.96324158 0.95495953 0.96631891 0.96875135]
Promedio R²: 0.96

Validación Cruzada Aleatoria (Shuffle Split)



Múltiples divisiones

Realiza varias particiones aleatorias del dataset, a diferencia del holdout que solo hace una.



Iteraciones

Para cada partición, se evalúa el modelo y se obtiene una métrica de rendimiento.



Promedio final

Se calculan los promedios de todas las evaluaciones para obtener una estimación más robusta.

La validación cruzada aleatoria o Shuffle Split es una alternativa flexible entre holdout y K-Fold. Es similar a realizar múltiples holdouts y promediar los resultados, lo que proporciona una evaluación más estable que un único holdout pero sin la estructura rígida del K-Fold.

```
# Importamos ShuffleSplit desde scikit-learn
# Esta técnica nos permite hacer validación cruzada con divisiones aleatorias del dataset
from sklearn.model_selection import ShuffleSplit

# Creamos una instancia de ShuffleSplit
# n_splits=10 → se harán 10 repeticiones (10 divisiones distintas del dataset)
# test_size=0.2 → en cada repetición, el 20% de los datos se usarán para prueba
# random_state=42 → garantiza que las divisiones sean siempre las mismas (para reproducibilidad)
ss = ShuffleSplit(n_splits=10, test_size=0.2, random_state=42)

# Aplicamos validación cruzada utilizando ShuffleSplit como estrategia de división
# cross_val_score entrenará el modelo 10 veces, cada vez con un conjunto distinto de datos
scores = cross_val_score(modelo, X, y, cv=ss)

# Mostramos el promedio de los 10 resultados obtenidos (por defecto usa scoring='r2')
print(f"Promedio (ShuffleSplit): {scores.mean():.2f}")
```

Promedio (ShuffleSplit): 0.97

Leave-One-Out (LOOCV)

1

Dato para prueba

En cada iteración, se deja un solo dato fuera para evaluar

N-1

Datos para entrenamiento

Se utilizan todos los demás datos para entrenar el modelo

N

Iteraciones totales

Se repite el proceso una vez por cada observación del dataset

Leave-One-Out Cross-Validation (LOOCV) es una forma especial de validación cruzada donde se usa una sola observación como conjunto de prueba en cada iteración, y el resto del dataset como conjunto de entrenamiento. Es considerado un caso extremo de K-Fold, donde K es igual al número total de observaciones.

Esta técnica es ideal para datasets pequeños donde cada muestra es valiosa, pero puede ser computacionalmente costosa en conjuntos de datos grandes.

```
[13]
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import LeaveOneOut, cross_val_score

# Generamos un dataset simple
X, y = make_regression(n_samples=20, n_features=1, noise=10, random_state=0)

# Creamos el modelo de regresión lineal
modelo = LinearRegression()

# Creamos el esquema LOOCV
loocv = LeaveOneOut()

# Usamos MSE como métrica (negativo por convención en sklearn)
scores = cross_val_score(modelo, X, y, cv=loocv, scoring='neg_mean_squared_error')

# Mostramos el promedio del error (convertido a positivo)
print(f"Promedio del error (LOOCV - MSE): {-scores.mean():.2f}")
```

➡ Promedio del error (LOOCV - MSE): 85.87

k-fold cross-validation

split showscrowing
olds and ifolds
ds and itrated
k times.

leave-one-out cross-validation

peating through
ch data point.

2. stratified k-fold cross-validation

Data split croavi
data split maint
class proprorpro
with sulte tands t
through each dat

4. time series cross-validation

time series-cho
etreiaring data
with forward-ch
data split

Comparativa de Técnicas de Validación

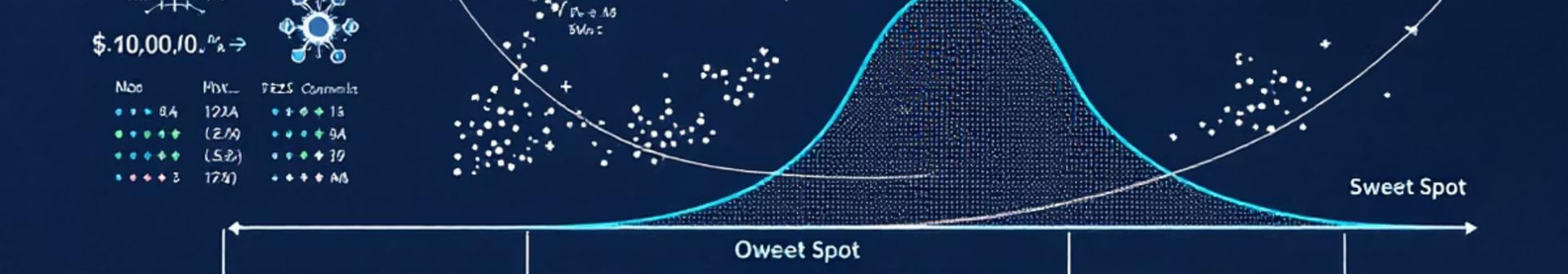
Técnica	¿Cómo funciona?	Ventajas principales	Desventajas principales
Holdout	Divide el dataset una sola vez (80/20)	Muy rápida y fácil	Inestable, depende de una única partición
K-Fold	Divide en K partes, entrena K veces	Más estable, usa todos los datos	Más costoso computacionalmente
ShuffleSplit	Genera múltiples divisiones aleatorias	Flexible y aleatorio	No garantiza que todos los datos se usen como test
LOOCV	Deja un dato fuera en cada iteración	Usa casi todos los datos para entrenar	Muy lento en datasets grandes

Recursos Adicionales

Para profundizar en los conceptos vistos en esta presentación, recomendamos revisar los siguientes recursos:

- SOBREAJUSTE Y SUBAJUSTE EN MACHINE LEARNING: Este video ofrece una explicación clara y concisa sobre cómo identificar y evitar el sobreajuste y subajuste en modelos de Machine Learning. [Ver en YouTube](#)
- ****Underfitting y Overfitting: Selección del Mejor Modelo | Clase**** : Una clase detallada que aborda la selección del modelo adecuado, explicando cómo el sobreajuste y subajuste afectan el rendimiento del modelo. [Ver en YouTube](#)
- Leer Artículo: Machine Bias: [Sesgos Inconscientes en el Machine Learning](#)
- Video: Machine Bias: [Sesgos inconscientes y machine learning](#)
- Leer Artículo: [El Sesgo y la Varianza en el Machine Learning - Mindful ML](#)
- Video: [BIAS Y VARIANZA EN MACHINE LEARNING | #10 Curso de Machine Learning](#)
- Video: [Evalúa un modelo de Machine Learning con validación cruzada | Machine Learning 101](#)

Este video explica cómo utilizar la validación cruzada para evaluar modelos de Machine Learning, destacando su importancia para evitar el sobreajuste.



Preguntas Clave sobre Modelado



Técnica de validación

¿Qué técnica de validación cruzada te pareció más útil o realista para aplicar en proyectos con datos reales? ¿Por qué?



Equilibrio del modelo

¿Qué aprendiste sobre el equilibrio entre un modelo que "sabe poco" y uno que "sabe demasiado"? ¿Cuál crees que es el mayor riesgo?



Rendimiento a largo plazo

¿Cómo te aseguras en el futuro de que tu modelo no solo funcione, sino que funcione bien a lo largo del tiempo?