SESIÓN NIVEL DE AJUSTE DE UN MODELO Y VALIDACIÓN CRUZADA

CONTENIDOS:

- Nivel de ajuste de un modelo: tipos de error de un modelo.
- Sobreajuste, subajuste y ajuste apropiado de un modelo.
- Trade-off entre sesgo (bias) y varianza.
- Validación cruzada: qué es validación cruzada:
    - Técnicas de validación cruzada.
    - Método de retención.
    - Validación cruzada de k-iteraciones.
    - Validación cruzada aleatoria.
    - Validación cruzada dejando uno afuera.
- Implementación de validación cruzada con la librería Scikit Learn.

NIVEL DE AJUSTE DE UN MODELO: TIPOS DE ERROR DE UN MODELO

El nivel de ajuste de un modelo hace referencia a la capacidad de un modelo para representar adecuadamente los datos observados. Revisemos los tipos de error en un modelo:

1. Error de Entrenamiento

El error de entrenamiento mide la diferencia entre las predicciones del modelo y los valores reales utilizando el conjunto de datos de entrenamiento. Este error es generalmente bajo porque el modelo ha sido ajustado específicamente para este conjunto de datos. Sin embargo, un error de entrenamiento bajo no garantiza un buen rendimiento en nuevos datos no vistos.

Para ejemplificar, imaginemos que estás desarrollando un modelo de aprendizaje automático para predecir la calificación de un estudiante basada en horas de estudio y asistencia a clases. Durante el entrenamiento del modelo, obtienes una predicción de 90% de precisión. Este es el **error de entrenamiento**, ya que refleja la diferencia entre las calificaciones predichas y las reales en el conjunto de datos de entrenamiento.

2. Error de Validación

El error de validación evalúa el rendimiento del modelo en un conjunto de datos de validación que no se usó durante el entrenamiento. Proporciona una estimación de cómo se comportará el modelo en datos no vistos y ayuda a identificar problemas de sobreajuste (overfitting). Un error de validación bajo sugiere que el modelo generaliza bien a nuevos datos.

Usando el mismo ejemplo y modelo anterior, decides validar su rendimiento en un conjunto de datos separado (validación) que no se utilizó durante el entrenamiento. Aquí, el modelo predice las calificaciones con un 85% de precisión. La diferencia entre las predicciones y las calificaciones reales en este conjunto es el **error de validación**.

3. Error de Prueba

El error de prueba se mide utilizando un conjunto de datos de prueba independiente, que se mantiene reservado hasta el final del proceso de modelado. Este error proporciona una evaluación final del rendimiento del modelo y asegura que no hubo optimización excesiva basada en el conjunto de validación.

Continuando con el ejemplo: Después de ajustar y validar tu modelo, evalúas su desempeño final en un conjunto de datos de prueba completamente nuevo, nunca antes visto por el modelo. El modelo predice las calificaciones con una precisión del 80%. Este es el **error de prueba**, y proporciona una estimación final del rendimiento del modelo en datos no vistos.

Tipo de Error	Descripción
Error de Entrenamiento	Diferencia entre las predicciones del modelo y los valores reales en el conjunto de datos de entrenamiento.
Error de Validación	Diferencia entre las predicciones del modelo y los valores reales en el conjunto de datos de validación.
Error de Prueba	Diferencia entre las predicciones del modelo y los valores reales en el conjunto de datos de prueba.

Ilustración 1 Tipos de error.

SOBREAJUSTE, SUBAJUSTE Y AJUSTE APROPIADO DE UN MODELO

Sobreajuste (Overfitting)

El sobreajuste ocurre cuando un modelo se ajusta demasiado bien a los datos de entrenamiento, capturando tanto patrones reales como ruido y anomalías. Esto resulta en un rendimiento pobre en datos nuevos. Un modelo sobreajustado puede tener un error de entrenamiento muy bajo, pero un error de validación o prueba alto.

Subajuste (Underfitting)

El subajuste ocurre cuando un modelo es demasiado simple para capturar los patrones subyacentes en los datos. Esto se traduce en un rendimiento pobre tanto en los datos de entrenamiento como en los de validación. Un modelo subajustado tiene tanto un error de entrenamiento alto como un error de validación alto.

Veamos algunos casos para ejemplificar: Supongamos que estás entrenando un modelo de clasificación para identificar especies de flores basándote en características como longitud de pétalo y ancho de sépalo. Si entrenas un modelo extremadamente complejo, como una red neuronal profunda con muchas capas, puede que el modelo se ajuste demasiado a los datos de entrenamiento, capturando ruido y variaciones específicas de esos datos. Esto resulta en un rendimiento excelente en el conjunto de entrenamiento, pero pobre en datos nuevos. Esto sería **Overfitting**

Ahora, si usas un modelo muy simple, como una regresión lineal para el mismo problema de clasificación de flores, puede que el modelo no capture las relaciones complejas entre las características y las especies. Esto resulta en un rendimiento pobre tanto en el conjunto de entrenamiento como en el conjunto de validación, porque el modelo es demasiado simple para capturar los patrones subyacentes. Aquí tenemos un problema de **Underfitting**

Ajuste Apropiado

El ajuste apropiado se logra cuando el modelo equilibra correctamente la complejidad y la capacidad de generalización. Capturan bien los patrones relevantes sin sobreajustarse ni subajustarse, proporcionando un rendimiento sólido tanto en los datos de entrenamiento como en los de validación.

Por ejemplo, si reutilizamos el problema de clasificación de flores, podrías optar por un modelo de complejidad intermedia, como un árbol de decisión. Este modelo puede capturar las relaciones relevantes entre las características sin sobreajustarse a las peculiaridades de los datos de entrenamiento, proporcionando un buen rendimiento en datos de validación y prueba, y por lo tanto un **Ajuste Apropiado.**

Concepto	Descripción
Sobreajuste	Ocurre cuando el modelo se ajusta demasiado a los datos de entrenamiento, capturando ruido y patrones no generalizables.
Subajuste	Ocurre cuando el modelo es demasiado simple para capturar los patrones subyacentes en los datos.
Ajuste Apropiado	El modelo captura bien los patrones relevantes en los datos sin sobreajustarse ni subajustarse.

Ilustración 2 Sobreajuste, subajuste y ajuste apropiado.

TRADE-OFF ENTRE SESGO (BIAS) Y VARIANZA

Sesgo (Bias)

El sesgo se refiere al error debido a las suposiciones simplificadas que hace el modelo. Un modelo con alto sesgo es demasiado simple y no captura los patrones subyacentes en los datos, resultando en subajuste.

En un problema de predicción de precios de viviendas, un modelo con alto sesgo podría ser una regresión lineal simple que asume que el precio de una casa depende linealmente solo de su tamaño. Esta suposición simplificada puede llevar a errores sistemáticos porque no captura otros factores importantes como la ubicación o el estado de la propiedad.

Varianza

La varianza se refiere a la sensibilidad del modelo a las pequeñas fluctuaciones en los datos de entrenamiento. Un modelo con alta varianza se ajusta demasiado a los datos de entrenamiento y tiene un rendimiento pobre en datos nuevos, resultando en sobreajuste.

Usando el mismo problema de predicción de precios de viviendas, un modelo con alta varianza podría ser una red neuronal profunda muy compleja. Este modelo puede ajustarse demasiado a las peculiaridades de los datos de entrenamiento y mostrar un rendimiento inconsistente en nuevos datos debido a su sensibilidad a pequeñas variaciones en los datos de entrenamiento.

Concepto	Descripción
Sesgo (Bias)	Error debido a suposiciones simplificadas en el modelo. Altos sesgos pueden llevar a subajuste.
Varianza	Error debido a la sensibilidad del modelo a pequeñas fluctuaciones en los datos de entrenamiento. Alta varianza puede llevar a sobreajuste.
Compensación Sesgo-Varianza	Encontrar el equilibrio adecuado entre sesgo y varianza para obtener un modelo con buen rendimiento en datos no vistos.

Ilustración 3 Bias y Varianza.

Compensación Sesgo-Varianza (Bias-Variance Trade-off)

El trade-off entre sesgo y varianza implica encontrar un equilibrio adecuado entre la simplicidad del modelo (bajo sesgo) y su capacidad para generalizar (baja varianza). El objetivo es diseñar un modelo con un buen rendimiento general, minimizando tanto el sesgo como la varianza.

Ejemplo Trade-off:

- **Modelo con Alto Sesgo:** Es como usar una línea recta para ajustar datos que claramente forman una curva. El modelo no captura la relación subyacente entre las variables.
- **Modelo con Alta Varianza:** Es como ajustar una curva muy compleja a unos pocos puntos de datos. El modelo capta el ruido y las fluctuaciones menores en los datos de entrenamiento, pero no generaliza bien a datos nuevos.

- **Modelo con Buen Equilibrio:** Ajusta una curva razonable que refleja la relación real en los datos sin captar ruido innecesario.

Tipo de Modelo	Descripción	Ejemplo Visual
Modelo con Alto Sesgo	Usa una línea recta para ajustar datos que forman una curva, no capturando la relación real.	<Predecir el precio de una casa basada solo en el tamaño sin considerar otras características.>
Modelo con Alta Varianza	Ajusta una curva muy compleja a unos pocos puntos de datos, capturando ruido y fluctuaciones menores.	<Red neuronal profunda que se ajusta demasiado a los datos de entrenamiento de precios de viviendas.>
Modelo con Buen Equilibrio	Ajusta una curva razonable que refleja la relación real en los datos sin captar ruido innecesario.	<Random forest que equilibra la complejidad y la capacidad de generalización en la predicción de precios de viviendas.>

Ilustración 4 Tipos modelos Trade-Off.

La clave es encontrar un modelo de complejidad adecuada que balancee el sesgo y la varianza. Un modelo como el **random forest** puede ser una buena opción para el problema de predicción de precios de viviendas. Este modelo utiliza múltiples árboles de decisión, lo que le permite capturar relaciones complejas (baja varianza) y evitar suposiciones simplificadas (bajo sesgo), proporcionando un rendimiento equilibrado en datos nuevos.

VALIDACIÓN CRUZADA: QUÉ ES LA VALIDACIÓN CRUZADA

La validación cruzada es una técnica utilizada para evaluar el rendimiento de un modelo de aprendizaje automático. Consiste en dividir los datos disponibles en varios subconjuntos y entrenar el modelo en algunos mientras se valida en otros. Esto se hace de manera iterativa, asegurando que cada subconjunto se use al menos una vez como conjunto de validación. La validación cruzada es una herramienta crucial para medir cómo de bien puede generalizar un modelo a datos no vistos, previniendo el sobreajuste.

La validación cruzada se basa en la idea de dividir el conjunto de datos disponible en múltiples subconjuntos para entrenar y evaluar el modelo de manera iterativa. En lugar de utilizar un único conjunto de entrenamiento y otro de prueba, la validación cruzada repite el proceso de entrenamiento y evaluación varias veces, utilizando diferentes particiones de los datos. Esto permite obtener una estimación más robusta y confiable del rendimiento del modelo.

Ilustración 5 Descripción grafica de validación cruzada.

Técnicas de validación cruzada

Existen varias técnicas de validación cruzada, cada una con sus propias características y aplicaciones. Las más comunes son:

1. **Método de Retención (Hold-Out Method)**:
    - Consiste en dividir el conjunto de datos en dos partes: un **conjunto de entrenamiento** (usado para ajustar el modelo) y un **conjunto de prueba** (usado para evaluar su rendimiento).
    - Es rápido y fácil de implementar, pero su estimación del rendimiento puede variar según cómo se dividan los datos.

- Es ideal para conjuntos de datos grandes o cuando se necesita una evaluación rápida, pero menos confiable en conjuntos pequeños o cuando se requiere una estimación más robusta.

2. **Validación Cruzada k-Fold (k-Fold Cross-Validation)**:
   - En este método, el conjunto de datos se divide en *k* subconjuntos (o "folds") de tamaño aproximadamente igual.
   - El modelo se entrena *k* veces, utilizando en cada iteración *k*-1 `folds` para entrenamiento y el `fold` restante para evaluación.
   - Finalmente, se calcula el promedio de las métricas de rendimiento obtenidas en cada iteración.
   - Este método es ampliamente utilizado porque proporciona un buen equilibrio entre sesgo y varianza en la estimación del rendimiento.
3. **Random Subsampling (Monte Carlo Cross-Validation)**:
   - Implica dividir repetidamente los datos en conjuntos de entrenamiento y validación de forma aleatoria y evaluar el modelo en varias iteraciones.
   - **Ventaja**: Permite una evaluación diversa y reduce el riesgo de divisiones desfavorables.
   - **Desventaja**: Puede ser menos representativa de la distribución completa de los datos si no se realizan suficientes iteraciones.
4. **Validación Cruzada Leave-One-Out (LOO)**:
   - Es un caso especial de k-Fold donde *k* es igual al número de muestras en el conjunto de datos.
   - En cada iteración, se deja fuera una única muestra para evaluación y se entrena el modelo con el resto de los datos.
   - Aunque este método es computacionalmente costoso, es útil cuando el conjunto de datos es muy pequeño, ya que maximiza la cantidad de datos utilizados para entrenamiento.

Método de retención

El método de retención (*hold-out method*) es una técnica simple y ampliamente usada en aprendizaje automático para evaluar modelos predictivos. Consiste en dividir el conjunto de datos en dos partes: un **conjunto de entrenamiento** (para ajustar el modelo) y un **conjunto de prueba** (para evaluar su rendimiento). La proporción típica es 80%-20%, aunque puede variar según el tamaño de los datos.

Pasos:

1. **División de datos**: Separar aleatoriamente los datos en entrenamiento y prueba.
2. **Entrenamiento**: Ajustar el modelo con el conjunto de entrenamiento.
3. **Evaluación**: Medir el rendimiento del modelo con el conjunto de prueba.
4. **Interpretación**: Si el rendimiento es satisfactorio, el modelo puede implementarse.

Ventajas:

- **Simplicidad**: Fácil de implementar y entender.
- **Bajo costo computacional**: Solo requiere un entrenamiento y una evaluación.
- **Efectivo en grandes conjuntos de datos**: El conjunto de prueba es representativo.

Limitaciones:

- **Dependencia de la partición**: El rendimiento puede variar según cómo se dividan los datos.
- **Uso ineficiente de datos**: Parte de los datos no se usa para entrenar.
- **Menos robusto**: Proporciona una única estimación del rendimiento.
- **No ideal para pequeños conjuntos**: El conjunto de prueba puede ser demasiado pequeño.

Comparación con Validación Cruzada:

El método de retención divide los datos en dos partes, mientras que la validación cruzada los divide en múltiples partes y realiza varias iteraciones.

Es preferible cuando los datos son grandes, se necesita rapidez o los recursos computacionales son limitados.

Aplicaciones:

Evaluación inicial: Útil para una estimación rápida del rendimiento.

Grandes conjuntos de datos: Adecuado cuando el conjunto de prueba es representativo.

Prototipado rápido: Ideal para probar modelos en etapas iniciales.

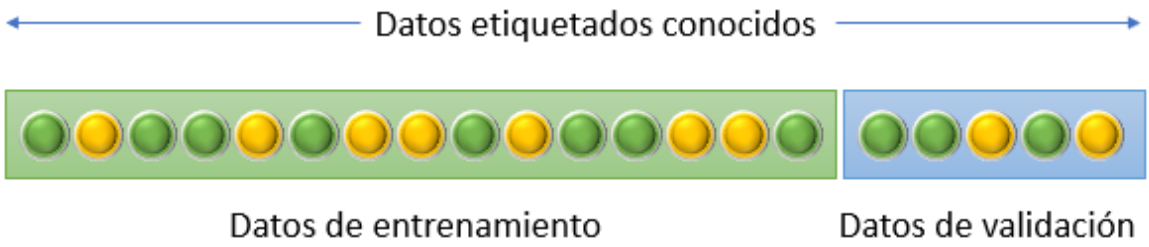

Ilustración 6 Método de Retención.

En resumen, el método de retención es una herramienta sencilla y eficiente para evaluar modelos, especialmente en grandes conjuntos de datos, aunque tiene limitaciones en términos de robustez y uso de datos.

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Cargar datos
iris = load_iris()
X, y = iris.data, iris.target

# Dividir los datos en entrenamiento y validación
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Modelo
model = DecisionTreeClassifier()

# Entrenar el modelo
model.fit(X_train, y_train)

# Validar el modelo
y_pred = model.predict(X_val)

# Calcular y mostrar la precisión
print("Accuracy:", accuracy_score(y_val, y_pred))
```

Ilustración 7 Script de Método de Retención.

K-Fold Cross-Validation

La **validación cruzada k-Fold (k-Fold Cross-Validation)** es una técnica robusta y ampliamente utilizada en aprendizaje automático para evaluar modelos predictivos. Consiste en dividir el conjunto de datos en *k* subconjuntos (o "folds") de tamaño similar. El modelo se entrena y evalúa *k* veces, utilizando en cada iteración *k*−1 `folds` para entrenamiento y el `fold` restante para evaluación. Finalmente, se calcula el promedio de las métricas de rendimiento obtenidas en cada iteración.

Pasos:

1. **División de datos**: Separar el conjunto de datos en *k* folds.
2. **Entrenamiento y evaluación**: En cada iteración, se entrena el modelo con *k*−1 `folds` y se evalúa con el `fold` restante.
3. **Cálculo del rendimiento**: Se promedian las métricas de rendimiento (precisión, exactitud, error cuadrático medio, etc.) de las *k* iteraciones.

Ventajas:

- **Robustez**: Proporciona una estimación más confiable del rendimiento al evaluar el modelo en múltiples particiones.
- **Uso eficiente de datos**: Todos los datos se utilizan tanto para entrenamiento como para evaluación.
- **Detección de sobreajuste**: Ayuda a identificar si el modelo generaliza bien o se ajusta demasiado a los datos.

Limitaciones:

- **Costo computacional**: Requiere entrenar y evaluar el modelo *k* veces, lo que puede ser costoso en tiempo y recursos.
- **Dependencia de *k***: La elección de *k* puede afectar la estimación del rendimiento (por ejemplo, *k*=10 es común, pero puede no ser óptimo en todos los casos).
- **No ideal para series temporales**: No respeta el orden temporal de los datos, lo que puede ser problemático en problemas de series de tiempo.

Comparación con el Método de Retención:

A diferencia del método de retención, que divide los datos en dos partes, la validación cruzada k-Fold utiliza múltiples particiones y promedia los resultados, lo que la hace más robusta.

Es preferible cuando se necesita una estimación precisa del rendimiento, especialmente en conjuntos de datos pequeños o medianos.

Aplicaciones:

- **Evaluación precisa**: Ideal para obtener una estimación confiable del rendimiento del modelo.
- **Conjuntos de datos pequeños**: Maximiza el uso de los datos, lo que es crucial cuando el conjunto de datos es limitado.
- **Selección de modelos**: Útil para comparar diferentes modelos y seleccionar el que mejor generaliza.

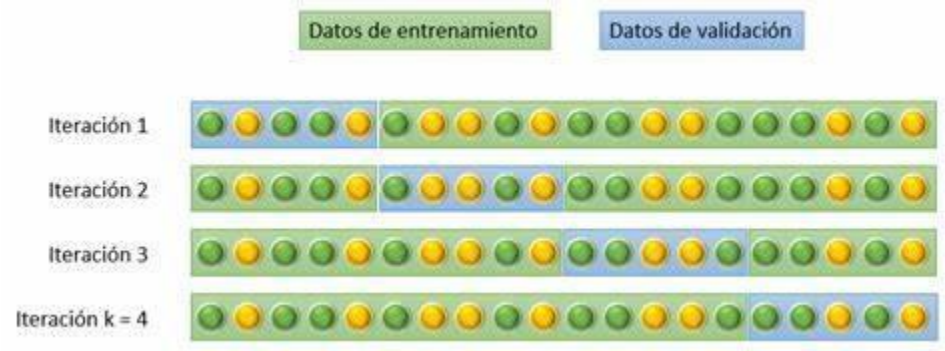

Ilustración 8 Validación cruzada de K iteraciones con K=4.

La validación cruzada k-Fold es una técnica poderosa para evaluar modelos, especialmente cuando se busca una estimación robusta del rendimiento. Sin embargo, su costo computacional y dependencia de *k* deben considerarse al aplicarla.

```python
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

# Cargar datos
iris = load_iris()
X, y = iris.data, iris.target

# Modelo
model = DecisionTreeClassifier()

# K-Fold Cross-Validation con 5 folds
kf = KFold(n_splits=5)

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print("Accuracy:", accuracy_score(y_test, y_pred))
```

Ilustración 9 Script de K-Fold Cross-Validation.

Random Subsampling (Monte Carlo Cross-Validation)

La **validación cruzada por submuestreo aleatorio (*Random Subsampling o Monte Carlo Cross-Validation*)** es una técnica flexible en aprendizaje automático para evaluar modelos predictivos. A diferencia de métodos como k-Fold o Leave-One-Out, este enfoque divide repetidamente el conjunto de datos en conjuntos de entrenamiento y prueba de manera aleatoria, sin una estructura fija. En cada iteración, se selecciona un subconjunto aleatorio de los datos para entrenamiento y el resto para prueba, repitiendo el proceso varias veces.

Pasos:

1. **División aleatoria**: En cada iteración, se selecciona un subconjunto aleatorio de los datos para entrenamiento y el resto para prueba.
2. **Entrenamiento y evaluación**: Se entrena el modelo con el subconjunto de entrenamiento y se evalúa con el subconjunto de prueba.

3. **Repetición**: El proceso se repite un número determinado de veces (definido por el usuario).
4. **Cálculo del rendimiento**: Se promedian las métricas de rendimiento obtenidas en todas las iteraciones.

Ventajas*:*

- **Flexibilidad**: Permite controlar el tamaño de los conjuntos de entrenamiento y prueba en cada iteración.
- **Evaluación diversa**: Al ser aleatorio, reduce el riesgo de sesgo debido a una partición específica de los datos.
- **Adaptabilidad**: Útil cuando no se quiere seguir una estructura fija, como en k-Fold.

Limitaciones*:*

- **Posible solapamiento**: Al ser aleatorio, algunas muestras pueden aparecer múltiples veces en el conjunto de prueba, lo que puede sesgar la evaluación.
- **Menos eficiente**: No garantiza que todas las muestras se usen para entrenamiento o prueba un número equilibrado de veces.
- **Dependencia del número de iteraciones**: Se requiere un número suficiente de iteraciones para obtener una estimación confiable, lo que puede aumentar el costo computacional.

Comparación con Otras Técnicas:

- A diferencia de k-Fold, que divide los datos en folds estructurados, Random Subsampling es completamente aleatorio, lo que puede ser más flexible pero menos sistemático.
- Comparado con Leave-One-Out, es menos costoso computacionalmente, pero puede ser menos preciso en conjuntos de datos pequeños.
- Frente al método de retención, ofrece una evaluación más robusta al repetir el proceso varias veces, pero con un mayor costo computacional.

Aplicaciones:

- **Evaluación flexible**: Ideal cuando se necesita controlar el tamaño de los conjuntos de entrenamiento y prueba.
- **Conjuntos de datos grandes**: Útil cuando el costo computacional de k-Fold o LOOCV es prohibitivo.

- **Prototipado rápido**: Adecuado para probar modelos en escenarios donde la aleatoriedad puede ayudar a identificar patrones inesperados.

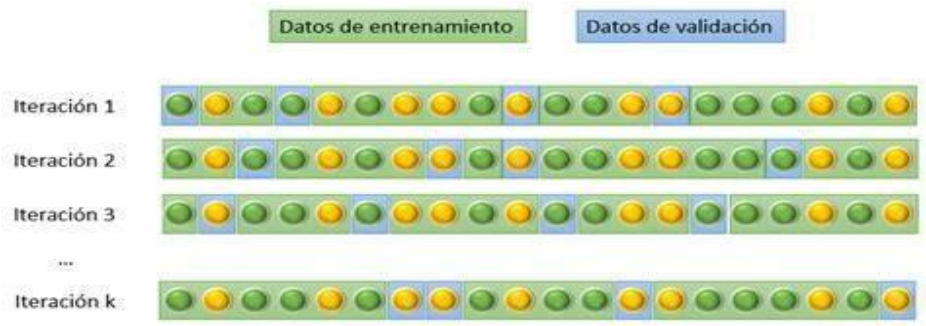

Ilustración 10 Validación Cruzada Aleatoria.

La validación cruzada por submuestreo aleatorio (Monte Carlo Cross-Validation) es una técnica versátil que ofrece flexibilidad y diversidad en la evaluación de modelos. Sin embargo, su naturaleza aleatoria puede introducir sesgos y requiere un número suficiente de iteraciones para ser confiable. Es especialmente útil en conjuntos de datos grandes o cuando se necesita una evaluación más dinámica.

```python
from sklearn.model_selection import ShuffleSplit
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Cargar datos
iris = load_iris()
X, y = iris.data, iris.target

# Modelo
model = DecisionTreeClassifier()

# Random Subsampling con 10 iteraciones
rs = ShuffleSplit(n_splits=10, test_size=0.2)

for train_index, test_index in rs.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print("Accuracy:", accuracy_score(y_test, y_pred))
```

Ilustración 11 Script de Random Subsampling.

Leave-One-Out Cross-Validation (LOOCV)

La **validación cruzada Leave-One-Out (LOOCV)** es una técnica especializada en aprendizaje automático para evaluar modelos predictivos. Es un caso extremo de la validación cruzada k-Fold, donde *k* es igual al número total de muestras en el conjunto de datos. En cada iteración, se deja fuera una única muestra para evaluación y se entrena el modelo con el resto de los datos. Este proceso se repite hasta que cada muestra ha sido utilizada exactamente una vez como conjunto de prueba.

Pasos*:*

1. **División de datos**: Se deja fuera una muestra y se entrena el modelo con $n-1$ muestras restantes.
2. **Evaluación**: Se evalúa el modelo con la muestra dejada fuera.
3. **Repetición**: El proceso se repite n veces (una por cada muestra).
4. **Cálculo del rendimiento**: Se promedian las métricas de rendimiento obtenidas en todas las iteraciones.

Ventajas*:*

- **Máximo uso de datos**: Cada iteración utiliza casi todos los datos para entrenamiento, lo que es ideal para conjuntos de datos muy pequeños.
- **Estimación precisa**: Proporciona una estimación casi insesgada del rendimiento del modelo, ya que se evalúa en cada muestra individual.
- **Sin dependencia de *k***: Al ser *k*=*n*, no hay necesidad de elegir un valor para *k*.

Limitaciones*:*

- **Alto costo computacional**: Requiere entrenar y evaluar el modelo n veces, lo que puede ser extremadamente costoso en conjuntos de datos grandes.
- **Varianza alta**: Debido a que cada iteración utiliza casi los mismos datos para entrenamiento, las estimaciones pueden tener una alta varianza.
- **No escalable**: No es práctico para conjuntos de datos grandes debido a su alto costo computacional.

Comparación con Otras Técnicas:

- A diferencia de la validación cruzada k-Fold, que divide los datos en *k* folds, LOOCV utiliza cada muestra individual como conjunto de prueba, lo que la hace más precisa pero menos eficiente.
- Comparado con el método de retención, LOOCV es más robusto y utiliza mejor los datos, pero es mucho más costoso computacionalmente.

Aplicaciones:

- **Conjuntos de datos pequeños**: Ideal cuando el número de muestras es limitado y se necesita maximizar el uso de los datos.
- **Precisión crítica**: Útil en aplicaciones donde se requiere una estimación muy precisa del rendimiento del modelo.
- **Investigación y prototipado**: Adecuado para estudios académicos o experimentales donde la precisión es más importante que la eficiencia.

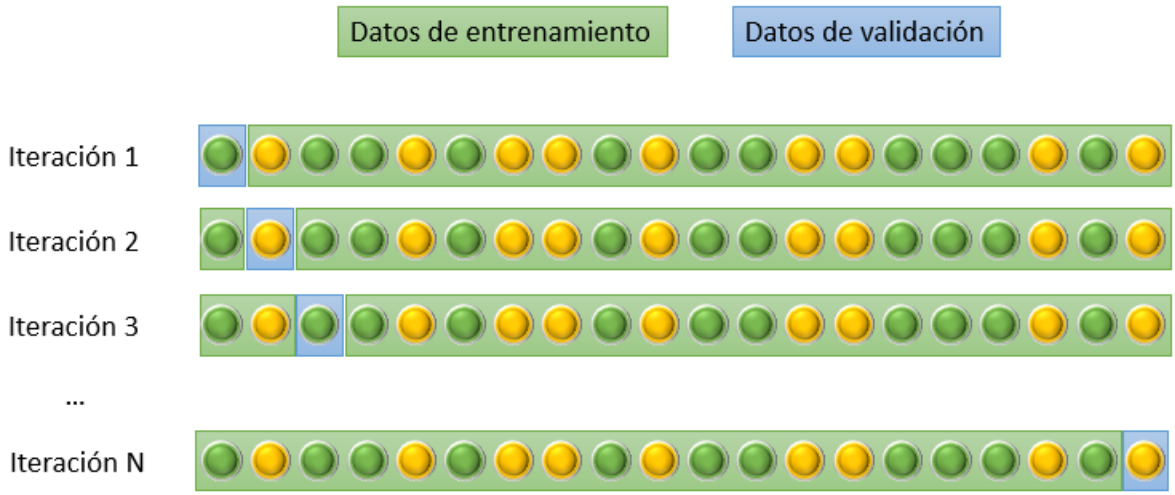

Ilustración 12 validación cruzada Leave-One-Out (LOOCV).

```python
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression

# Datos de ejemplo
X = [[1, 2], [2, 4], [3, 6], [4, 8], [5, 10]]
y = [1, 2, 3, 4, 5]

# Modelo
model = LinearRegression()

# Leave-One-Out Cross-Validation
loo = LeaveOneOut()
mse_scores = []

for train_index, test_index in loo.split(X):
    X_train, X_test = [X[i] for i in train_index], [X[i] for i in test_index]
    y_train, y_test = [y[i] for i in train_index], [y[i] for i in test_index]
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse_scores.append(mean_squared_error(y_test, y_pred))

print("Mean MSE:", sum(mse_scores) / len(mse_scores))
```

Ilustración 13 Script de LOOCV.

IMPLEMENTACIÓN DE VALIDACIÓN CRUZADA CON LA LIBRERÍA SCIKIT-LEARN

La **implementación de validación cruzada con la librería Scikit-Learn** es un proceso sencillo y eficiente, gracias a las herramientas integradas que ofrece esta popular biblioteca de Python para aprendizaje automático. Scikit-Learn proporciona funciones y clases específicas para aplicar diversas técnicas de validación cruzada, como **k-Fold**, **Leave-One-Out (LOOCV)** y **Random Subsampling**, entre otras. A continuación, haremos un **ejercicio guiado** de cómo implementar estas técnicas paso a paso.

ACTIVIDAD PRÁCTICA GUIDA:

Paso 0: Instalación de Scikit-Learn

Antes de comenzar, asegúrate de tener instalada la librería Scikit-Learn. Si no la tienes, puedes instalarla usando pip. Idealmente ocupar venv

```
PS C:\> python -m venv sklearn-env
PS C:\> sklearn-env\Scripts\activate  # activate
PS C:\> pip install -U scikit-learn
```

Ilustración 14 instalación scikit-Learn con venv.

Paso 1: Importación de Librerías

Primero, importamos las librerías necesarias para trabajar con datos y modelos de aprendizaje automático.

```
# Importar librerías
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import (
    KFold,
    LeaveOneOut,
    StratifiedKFold,
    ShuffleSplit,
    cross_val_score,
)
from sklearn.ensemble import RandomForestClassifier
```

Ilustración 15 Script importar librerías.

Paso 2: Carga de Datos

Cargamos el conjunto de datos Iris, que es un dataset clásico para clasificación.

```
# Cargar el dataset Iris
data = load_iris()
X = data.data  # Características (features)
y = data.target  # Etiquetas (labels)
```

Ilustración 16 Script carga de datos.

Paso 3: Implementación de k-Fold Cross-Validation

Dividimos los datos en 5 folds y evaluamos el modelo en cada uno de ellos.

```
18  # Configurar k-Fold Cross-Validation
19  kf = KFold(n_splits=5, shuffle=True, random_state=42)
20
21  # Evaluar el modelo con k-Fold
22  model = RandomForestClassifier(random_state=42)
23  scores = cross_val_score(model, X, y, cv=kf)
24
25  # Mostrar los resultados
26  print("Puntuaciones de k-Fold Cross-Validation:", scores)
27  print("Puntuación media:", np.mean(scores))
```

Ilustración 17 Script k-Fold

Resultado Esperado:

```
Puntuaciones de k-Fold Cross-Validation: [1.         0.96666667 0.93333333 0.93333333 0.96666667]
Puntuación media: 0.9600000000000002
```

Ilustración 18 Resultado k-Fold

Paso 4: Implementación de Leave-One-Out Cross-Validation (LOOCV)

LOOCV es un caso especial de k-Fold donde k es igual al número de muestras.

```
29  # Configurar Leave-One-Out Cross-Validation
30  loo = LeaveOneOut()
31
32  # Evaluar el modelo con LOOCV
33  scores = cross_val_score(model, X, y, cv=loo)
34
35  # Mostrar los resultados
36  print("Puntuaciones de LOOCV:", scores)
37  print("Puntuación media:", np.mean(scores))
```

Ilustración 19 Script LOOCV

Resultado Esperado:

```
Puntuaciones de LOOCV: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
Puntuación media: 0.9533333333333334
```

Ilustración 20 Resultado LOOCV

Paso 5: Implementación de Random Subsampling (Monte Carlo Cross-Validation)

Random Subsampling no está directamente implementado en Scikit-Learn, pero puede simularse con ShuffleSplit.

```
39  # Configurar Random Subsampling
40  rs = ShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
41
42  # Evaluar el modelo con Random Subsampling
43  scores = cross_val_score(model, X, y, cv=rs)
44
45  # Mostrar los resultados
46  print("Puntuaciones de Random Subsampling:", scores)
47  print("Puntuación media:", np.mean(scores))
```

Ilustración 21 Script Random Subsampling.

Resultado Esperado:

```
Puntuaciones de Random Subsampling: [1.         0.96666667 0.96666667 0.93333333 0.93333333 1.
 0.9        0.96666667 1.         0.93333333]
Puntuación media: 0.96
```

Ilustración 22 Resultado Random Subsampling.

Paso 7: Comparación de Métodos

Finalmente, comparamos los resultados de las diferentes técnicas de validación cruzada.

```
49  # Comparar resultados
50  print("Comparación de métodos:")
51  print("k-Fold:", np.mean(cross_val_score(model, X, y, cv=KFold(n_splits=5, shuffle=True, random_state=42))))
52  print("LOOCV:", np.mean(cross_val_score(model, X, y, cv=LeaveOneOut())))
53  print("Random Subsampling:", np.mean(cross_val_score(model, X, y, cv=ShuffleSplit(n_splits=10, test_size=0.
    2, random_state=42))))
```

Ilustración 23 Script comparación.

Explicación:

- Este paso resume las puntuaciones medias de cada técnica para comparar su rendimiento.

Resultado Esperado:

```
Comparación de métodos:
k-Fold: 0.9600000000000002
LOOCV: 0.9533333333333334
Random Subsampling: 0.96
```

Ilustración 24 Resultado comparación.