

cross validation v2 - IBM



cross-validation-v2-ibm-machine-1.ipynb
File

Gracias por subir el archivo **cross-validation-v2-ibm-machine-1.ipynb**. Procederé a analizar todo el código del notebook celda por celda, explicando:

1. Qué hace cada bloque de código.
2. Los fundamentos teóricos detrás de cada instrucción.
3. Una conclusión global al final.
4. Teoría adicional y conceptos matemáticos relevantes.

Comienzo ahora con la primera celda. Apenas termine su explicación, seguiré con la siguiente. Vamos paso a paso.

◆ Celda 1

python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

✓ Explicación del código

Esta celda importa las bibliotecas principales utilizadas en análisis de datos y visualización en Python:

- **pandas**: Para manipulación y análisis de estructuras de datos, especialmente **DataFrames**.
- **numpy**: Biblioteca para operaciones numéricas eficientes, especialmente en arrays.
- **matplotlib.pyplot**: Herramienta fundamental para crear gráficos estáticos en 2D.
- **seaborn**: Basada en **matplotlib**, ofrece visualizaciones estadísticas de alto nivel con estilo más atractivo.

📚 Fundamentos teóricos

- **Pandas** es esencial para leer datos desde fuentes como CSV y manipular columnas, valores nulos, etc.

- **NumPy** ofrece estructuras de datos (**ndarray**) más eficientes que las listas nativas y operaciones vectorizadas.
- **Matplotlib** se basa en el paradigma de estilo MATLAB para gráficos (líneas, barras, dispersión, etc.).
- **Seaborn** facilita el análisis exploratorio visual de datos al incorporar funciones estadísticas como `sns.boxplot`, `sns.heatmap`, etc.

¿Quieres que continúe con la siguiente celda ahora?



continuar

Perfecto, continúo con la siguiente celda.

♦ Celda 2

python

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

✓ Explicación del código

Aquí se importan herramientas del módulo **sklearn** (scikit-learn), una de las bibliotecas más usadas para Machine Learning en Python:

- **train_test_split**: Para dividir los datos en conjunto de entrenamiento y prueba.
- **LinearRegression**: Modelo de regresión lineal simple/múltiple.
- **mean_squared_error**: Métrica que calcula el error cuadrático medio.
- **r2_score**: Métrica que devuelve el coeficiente de determinación R^2 .

Fundamentos teóricos

- **train_test_split** divide los datos aleatoriamente (por defecto en 75% train y 25% test). Esto es esencial para validar si un modelo generaliza bien.
- **LinearRegression** implementa el modelo:

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n + \varepsilon$$

donde los coeficientes β_i se estiman mediante **mínimos cuadrados ordinarios** (OLS).

- **mean_squared_error** (MSE) mide el promedio de los errores al cuadrado:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- `r2_score` mide la proporción de la varianza explicada por el modelo:

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

Un valor cercano a 1 indica buen ajuste; cercano a 0 o negativo indica mal ajuste.

¿Sigo con la próxima celda?



continuar

Perfecto, avanzamos con la siguiente celda.

◆ Celda 3

python

```
df = pd.read_csv("house.csv")
df.head()
```

✓ Explicación del código

- `pd.read_csv("house.csv")`: Carga un archivo CSV llamado `"house.csv"` en un DataFrame llamado `df`.
- `df.head()`: Muestra las primeras 5 filas del DataFrame para tener una vista rápida de los datos.

📚 Fundamentos teóricos

- **Lectura de datos:** `read_csv` es una función fundamental en **pandas** para importar datasets en formato tabular. Es el primer paso del flujo de trabajo de ciencia de datos: *cargar* → *explorar* → *limpiar* → *modelar* → *evaluar*.
- `head()` permite hacer una inspección preliminar para:
 - Verificar que los datos fueron cargados correctamente.
 - Entender la estructura y nombres de columnas.
 - Detectar posibles valores nulos o tipos de datos inesperados.

¿Paso a la siguiente celda?



continuar

Seguimos con la siguiente celda.

◆ Celda 4

```
python

sns.regplot(x="area", y="price", data=df)
plt.show()
```

✓ Explicación del código

- `sns.regplot(...)`: Crea un gráfico de dispersión (*scatter plot*) con una línea de regresión ajustada.
- `x="area"`: Usa la columna `area` como variable independiente.
- `y="price"`: Usa la columna `price` como variable dependiente.
- `data=df`: Indica que las columnas provienen del DataFrame `df`.
- `plt.show()`: Muestra el gráfico generado.

📚 Fundamentos teóricos

- **Gráfico de dispersión (scatter plot)**: Permite visualizar la relación entre dos variables numéricas. Aquí: cuánto influye el área en el precio de una casa.
- **Línea de regresión**: Representa el mejor ajuste lineal entre `area` y `price` usando mínimos cuadrados. La pendiente de esta línea da una idea del efecto marginal de un aumento en el área sobre el precio.
- Esta visualización es muy útil para verificar visualmente:
 - Si hay una relación lineal (o no).
 - La presencia de posibles *outliers*.
 - Homocedasticidad (igual dispersión del error en todos los niveles de `x`).

¿Paso a la siguiente celda?



continuar

Perfecto, continuamos con la siguiente celda.

◆ Celda 5

```
python

X = df[["area"]]
y = df["price"]
```

✓ Explicación del código

- `x = df[["area"]]`: Extrae la columna **area** como variable independiente (input feature) y la guarda en un DataFrame `x`.
- `y = df["price"]`: Extrae la columna **price** como variable dependiente (target) y la guarda como **Series** `y`.

Notar que `x` se mantiene como un **DataFrame** (dobles corchetes), lo cual es importante para scikit-learn, ya que espera una matriz 2D para las variables de entrada.

Fundamentos teóricos

- En aprendizaje supervisado, se define:
 - $\mathbf{X} \in \mathbb{R}^{n \times m}$: matriz de características con n ejemplos y m variables.
 - $\mathbf{y} \in \mathbb{R}^n$: vector de etiquetas o respuestas reales.
- En este caso:
 - $m = 1$ porque sólo usamos **area** como input.
 - y representa el valor monetario de la propiedad.
- Este paso forma parte de la **separación de características y objetivo**, una práctica estándar antes de entrenar modelos.

¿Deseás que continúe con la siguiente celda?



continuar

Muy bien, seguimos con la siguiente celda.

◆ Celda 6

```
python

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

✓ Explicación del código

Este código divide el dataset en conjuntos de entrenamiento y prueba:

- **X_train, y_train**: Datos para entrenar el modelo (70% del total).
- **X_test, y_test**: Datos para evaluar el modelo (30% del total).

Parámetros clave:

- **test_size=0.3**: Usa el 30% de los datos para testing.

- `random_state=0`: Fija una semilla para obtener la misma partición cada vez (reproducibilidad).

Fundamentos teóricos

- **Partición de datos** es fundamental para evitar sobreajuste (*overfitting*). Entrenar y evaluar sobre los mismos datos lleva a resultados engañosamente optimistas.
- **Conjunto de entrenamiento**: Se usa para ajustar los parámetros del modelo (por ejemplo, coeficientes de regresión).
- **Conjunto de prueba**: Se utiliza para estimar cómo se desempeñaría el modelo con datos nuevos y no vistos.
- `random_state`: Asegura que la división aleatoria sea reproducible para que los experimentos sean consistentes.

Este es el primer paso antes de ajustar cualquier modelo predictivo supervisado.

¿Avanzamos con la siguiente celda?



continuar

Perfecto, continuamos con la siguiente celda.

◆ Celda 7

python

```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

✓ Explicación del código

- `lr = LinearRegression()`: Crea una instancia del modelo de regresión lineal de scikit-learn.
- `lr.fit(X_train, y_train)`: Ajusta el modelo a los datos de entrenamiento. Es decir, encuentra los parámetros (pendiente y ordenada al origen) que mejor se ajustan a los datos.

Fundamentos teóricos

Este bloque ejecuta el **entrenamiento del modelo de regresión lineal**, cuyo objetivo es encontrar una función lineal que minimice el error cuadrático medio:

$$\hat{y} = \beta_0 + \beta_1 x$$

- **Ajuste por mínimos cuadrados**: Busca los coeficientes β_0 (intercepto) y β_1 (pendiente) que minimicen:

$$\min_{\beta_0, \beta_1} \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 x_i))^2$$

- El método `fit()` calcula internamente estos parámetros usando álgebra matricial:

$$\beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

donde \mathbf{X} es la matriz de diseño (incluyendo columna de unos si hay intercepto).

Este paso es el **entrenamiento efectivo** del modelo.

¿Quieres que continúe con la próxima celda?



continuar

Perfecto, avancemos con la siguiente celda.

♦ Celda 8

```
python

yhat = lr.predict(X_test)
```

✓ Explicación del código

- `lr.predict(X_test)`: Usa el modelo entrenado para predecir los precios (**yhat**) en base a las áreas del conjunto de prueba (`X_test`).

📖 Fundamentos teóricos

- Este es el paso de **predicción** con un modelo previamente entrenado.
- Dado un nuevo valor x (área de una casa), el modelo devuelve:

$$\hat{y} = \beta_0 + \beta_1 x$$

- En este caso, para todos los valores de `X_test`, se genera un vector **yhat** con los valores estimados de precios.

Esta etapa sirve para comparar los valores **predichos** con los valores **reales** (`y_test`) y así evaluar el rendimiento del modelo.

¿Avanzo con la próxima celda?



continuar

Excelente, pasemos a la siguiente celda.

◆ Celda 9

python

```
plt.scatter(X_test, y_test, color="blue", label="Actual")
plt.plot(X_test, yhat, color="red", linewidth=2, label="Predicted")
plt.xlabel("Area")
plt.ylabel("Price")
plt.legend()
plt.show()
```

✓ Explicación del código

Esta celda genera una visualización comparando los valores reales vs. los predichos por el modelo:

- `plt.scatter(...)`: Dibuja un diagrama de dispersión con los valores reales (`X_test`, `y_test`).
- `plt.plot(...)`: Dibuja la línea de regresión obtenida a partir de las predicciones (`X_test`, `yhat`).
- `color`: Azul para valores reales, rojo para la predicción.
- `label`: Identificadores para mostrar en la leyenda.
- `xlabel` / `ylabel`: Etiquetas de los ejes.
- `plt.legend()`: Activa la leyenda.
- `plt.show()`: Muestra el gráfico final.

📚 Fundamentos teóricos

- Este gráfico es crucial para **evaluar visualmente** si el modelo sigue bien la tendencia de los datos reales.
- Una buena regresión debería tener una línea roja que pase lo más cerca posible de los puntos azules.

Este gráfico ayuda a detectar:

- Tendencias no lineales no captadas por el modelo.
- Outliers.
- Dispersión del error.

Es un **complemento visual** a las métricas numéricas como R^2 o el MSE.

¿Quieres que continúe con la próxima celda?



continuar

Perfecto, continuamos con la siguiente celda.

◆ Celda 10

python

```
print("Mean squared error (MSE): %.2f" % mean_squared_error(y_test, yhat))
print("Coefficient of determination (R^2): %.2f" % r2_score(y_test, yhat))
```

✓ Explicación del código

Esta celda imprime dos métricas clave para evaluar el rendimiento del modelo:

1. **Mean Squared Error (MSE):** Promedio del cuadrado de los errores entre los valores reales y predichos.
2. **R² (coeficiente de determinación):** Cuantifica qué proporción de la varianza de y es explicada por el modelo.

Ejemplo de salida:

java

```
Mean squared error (MSE): 2500.17
Coefficient of determination (R^2): 0.82
```

📖 Fundamentos teóricos

- **MSE:**

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Penaliza fuertemente los errores grandes.
- Cuanto más bajo, mejor.
- R^2 :

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

- Mide la proporción de la varianza explicada.
- $R^2 = 1$: predicción perfecta.
- $R^2 = 0$: el modelo no explica nada mejor que la media.
- Puede ser negativo si el modelo es peor que simplemente predecir la media.

Estas dos métricas se complementan:

- **MSE** nos dice *cuánto* se equivoca.
- R^2 nos dice *qué tan útil* es el modelo.

¿Quieres que siga con la próxima celda?



continuar

Seguimos con la próxima celda.

◆ Celda 11

```
python

from sklearn.model_selection import cross_val_score
```

✓ Explicación del código

- Se importa la función `cross_val_score` del módulo `sklearn.model_selection`.

Esta función permite realizar **validación cruzada** (cross-validation), una técnica fundamental para evaluar el rendimiento de un modelo de manera más robusta.

📖 Fundamentos teóricos

- **Validación cruzada** es un procedimiento que divide los datos en múltiples subconjuntos llamados *folds* (pliegues), y entrena/evalúa el modelo varias veces, rotando qué parte es entrenamiento y cuál es prueba.
- Por ejemplo, en **K-fold cross-validation** con $K = 5$, el conjunto de datos se divide en 5 partes. Cada vez se entrena en 4 y se prueba en 1, hasta cubrir todas las combinaciones.
Esto permite estimar mejor el rendimiento del modelo en datos no vistos y evita que los resultados dependan de una única división.
- `cross_val_score(...)` devuelve un array con las métricas (por defecto, el R^2) obtenidas en cada partición.

Esta función es clave para evitar **overfitting** y obtener una evaluación más **confiable** del modelo.

¿Pasamos a la próxima celda donde se utiliza esta función?



continuar

Perfecto, continuamos con la siguiente celda.

◆ Celda 12

```
python

scores = cross_val_score(lr, X, y, cv=5)
print("Cross-validation scores:", scores)
print("Average score: %.2f" % scores.mean())
```

✓ Explicación del código

- `cross_val_score(lr, X, y, cv=5)`: Ejecuta validación cruzada de 5 pliegues (folds) usando el modelo `lr` (regresión lineal), con los datos `X` e `y`.
- `scores`: Contiene 5 valores (uno por fold) del coeficiente R^2 obtenido en cada partición.
- `scores.mean()`: Calcula el promedio de esos valores.

📖 Fundamentos teóricos

- Se usa **K-Fold Cross-Validation** con $K = 5$. Cada fold contiene aproximadamente el 20% del dataset, rotando qué fold se usa como prueba.

El modelo se entrena en el 80% restante y se evalúa en el fold de prueba.

- Este proceso se repite 5 veces y se reportan 5 métricas de rendimiento. El promedio da una **estimación robusta del rendimiento general del modelo**.
- Ventajas clave:
 - Se reduce la varianza respecto a una sola partición train/test.
 - Se detecta si el modelo es sensible a los datos de entrada (inestable).
 - Aporta mayor confianza al evaluar modelos comparativos.

Esta es la forma estándar de evaluar modelos antes de seleccionar el mejor.

¿Pasamos a la próxima celda?



continuar

Perfecto, vamos con la siguiente celda.

◆ Celda 13

```
python

from sklearn.model_selection import KFold
```

✓ Explicación del código

- Esta línea importa la clase `KFold` desde `sklearn.model_selection`.

`KFold` es una herramienta para realizar validación cruzada dividiendo explícitamente el conjunto de datos en k particiones (folds), de manera más personalizada que `cross_val_score`.

📖 Fundamentos teóricos

- `KFold` es una clase que permite:
 - Dividir el dataset en k folds manualmente.
 - Controlar si los datos se mezclan (`shuffle`) antes de dividirlos.
 - Obtener los índices de entrenamiento y prueba para cada fold.

Esto permite una mayor flexibilidad sobre `cross_val_score`, por ejemplo:

- Obtener los valores predichos fold por fold.
- Evaluar con métricas personalizadas.
- Usar modelos que requieren entrenamiento más complejo o logging detallado.

Ideal cuando se necesita control total sobre el proceso de entrenamiento/evaluación fold-a-fold.

¿Continuamos con la celda donde se utiliza `KFold` explícitamente?



continuar

Muy bien, avancemos con la siguiente celda donde se aplica `KFold`.

◆ Celda 14

```
python

kf = KFold(n_splits=5)
kf.get_n_splits(X)
```

✓ Explicación del código

- `kf = KFold(n_splits=5)`: Crea una instancia de validación cruzada K-Fold con 5 particiones.
- `kf.get_n_splits(X)`: Devuelve el número de particiones que se generarán (aquí, 5).

Aunque `get_n_splits(X)` se muestra, su resultado no se asigna ni imprime, por lo tanto no afecta directamente el flujo.

Fundamentos teóricos

- El objeto `KFold` genera índices para dividir los datos en $K = 5$ folds.
- A diferencia de `cross_val_score`, aquí se debe implementar manualmente el ciclo de entrenamiento y evaluación, lo que da mayor control.
- Por defecto:
 - **No hay mezcla aleatoria** de los datos (`shuffle=False`).
 - Los folds son estratificados solo si se usa `StratifiedKFold` (para clasificación). Para regresión, `KFold` divide de manera secuencial.

Este paso simplemente **configura la estructura** de los folds, sin aún dividir ni evaluar.

¿Avanzo con la próxima celda donde se ejecuta el ciclo manual de entrenamiento usando `KFold`?



continuar

Perfecto, vamos ahora con la siguiente celda donde se implementa explícitamente la validación cruzada manual con `KFold`.

◆ Celda 15

python

```
for train_index, test_index in kf.split(X):
    print("Train:", train_index, "Validation:", test_index)
```

✓ Explicación del código

Este bucle recorre los 5 folds generados por el objeto `kf`:

- `kf.split(X)`: Devuelve tuplas con índices de entrenamiento y validación para cada fold.
- Dentro del `for`, se imprimen los índices que corresponden a los conjuntos de entrenamiento y validación en cada iteración.

Fundamentos teóricos

Este paso es útil para **visualizar cómo se divide el dataset** en cada iteración del K-Fold:

- Cada fold actúa una vez como conjunto de validación y $K - 1$ veces como conjunto de entrenamiento.
- En cada iteración:
 - **train_index**: índices que pertenecen al conjunto de entrenamiento.
 - **test_index**: índices que se dejan fuera para validación.

Aunque este bloque no entrena el modelo, es un paso **pedagógico y exploratorio**, para entender la partición de los datos. En práctica, luego se usarían estos índices para entrenar y evaluar el modelo manualmente.

¿Querés que continúe con la siguiente celda?



continuar

Perfecto, seguimos con la siguiente celda.

◆ Celda 16

```
python

from sklearn.model_selection import cross_val_predict
```

✓ Explicación del código

Se importa la función `cross_val_predict` desde `sklearn.model_selection`.

A diferencia de `cross_val_score`, esta función **devuelve las predicciones completas** obtenidas por validación cruzada, en lugar de solo las métricas.

📖 Fundamentos teóricos

- `cross_val_predict`:
 - Realiza K-Fold Cross-Validation internamente.
 - Para cada fold, entrena el modelo con los datos de entrenamiento.
 - Predice los valores del conjunto de prueba (fold) y los guarda.
 - Finalmente, devuelve **una predicción para cada instancia del conjunto completo**, como si cada una hubiera sido "vista por el modelo" en un fold donde no participó del entrenamiento.

Esto permite:

- Comparar directamente las predicciones generadas por validación cruzada contra los valores reales.
- Calcular métricas globales como MSE, R^2 , MAE, etc., sobre todas las predicciones simulando predicción real en datos no vistos.

Es ideal para diagnóstico de errores y evaluación sin *data leakage*.

¿Paso a la siguiente celda donde se utiliza esta función?



continuar

Perfecto, pasamos a la última celda del notebook.

◆ Celda 17

```
python

predicted = cross_val_predict(lr, X, y, cv=5)

plt.scatter(y, predicted, edgecolors=(0, 0, 0))
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Cross-Validated Predictions")
plt.show()
```

✓ Explicación del código

1. `predicted = cross_val_predict(...)`: Genera las predicciones para todos los datos usando validación cruzada de 5 folds, con el modelo de regresión lineal `lr`.
2. `plt.scatter(y, predicted, ...)`: Grafica los valores reales (`y`) vs. los predichos (`predicted`). Cada punto representa una observación.
3. `plt.plot(...)`: Dibuja una línea diagonal que representa una predicción perfecta (donde `Predicted = Actual`).
4. `plt.xlabel(...)`, `plt.ylabel(...)`, `plt.title(...)`: Etiquetas y título del gráfico.
5. `plt.show()`: Muestra el gráfico final.

Fundamentos teóricos

- Este gráfico se conoce como **gráfico de dispersión real vs. predicho**.
- Permite visualizar cuán cerca están las predicciones del modelo respecto a los valores reales.
 - **Línea diagonal (negra punteada)**:
 - Representa el caso ideal en que la predicción es igual al valor real.
 - Cuanto más cerca están los puntos de esta línea, mejor el modelo.
- `cross_val_predict` garantiza que **cada predicción fue hecha usando un modelo que no "vio" ese dato durante el entrenamiento**, simulando una verdadera generalización.

Es una de las herramientas más útiles para diagnosticar desempeño de modelos de regresión en validación cruzada.

✓ CONCLUSIÓN DETALLADA DEL NOTEBOOK

Este notebook es una implementación clara y pedagógica del uso de **regresión lineal** en conjunto con distintas técnicas de **evaluación**, destacando especialmente la **validación cruzada**. Aquí están los pasos

clave:

1. **Importación y carga de datos** (`house.csv`).
2. **Visualización exploratoria** para detectar relación lineal entre `area` y `price`.
3. **Separación de variables** independientes y dependientes.
4. **División en entrenamiento y prueba** con `train_test_split`.
5. **Entrenamiento del modelo** de regresión lineal con `LinearRegression`.
6. **Evaluación inicial** con MSE y R^2 .
7. **Aplicación de K-Fold Cross-Validation** mediante `cross_val_score`, `KFold`, y `cross_val_predict`.
8. **Visualización detallada** del desempeño de las predicciones validadas.

El objetivo del notebook no es solo ajustar un modelo, sino mostrar cómo evaluar correctamente su rendimiento y robustez ante nuevas particiones de los datos, usando validación cruzada.

TEORÍA Y CONCEPTOS ADICIONALES

Aquí te dejo una síntesis de conceptos fundamentales que podrías reforzar o profundizar:

♦ Regresión lineal (OLS)

- Técnica supervisada para modelar relaciones lineales.
- Supone normalidad de errores, homocedasticidad, no multicolinealidad, independencia.
- Puede extenderse a regresión múltiple, polinomial, regularizada (Ridge, Lasso).

♦ Validación cruzada

- Técnica de evaluación confiable.
- Evita *overfitting* al no depender de una sola división.
- Puede combinarse con búsqueda de hiperparámetros (`GridSearchCV`).

♦ Métricas de regresión

- **MSE**: Penaliza errores grandes.
- **MAE** (no usada aquí): Media del valor absoluto del error.
- R^2 : Medida relativa de varianza explicada.

♦ Visualizaciones clave

- `regplot`: Permite ver la relación lineal esperada.
- `real vs. predicted`: Diagnóstico visual excelente para regresión.

 **NO HAY MÁS CELDAS DE CÓDIGO PARA EXPLICAR.**

¿Querés que prepare un resumen en PDF, lo convierta en presentación o te ayude a replicarlo con tus propios datos?