

regularizaci-n-ibm-machine-l



regularizaci-n-ibm-machine-l.ipynb
File

Gracias por subir el archivo **regularizaci-n-ibm-machine-l.ipynb**. Procederé ahora a analizar profesionalmente todo el código celda por celda, explicando:

1. Cada línea y bloque de código
2. Los fundamentos teóricos y matemáticos
3. Conclusión detallada al finalizar el análisis
4. Teoría adicional relacionada con regularización y aprendizaje automático

Comienzo ahora a leer y explicarte el contenido del notebook. Un momento...

✅ EXPLICACIÓN DETALLADA DE LAS PRIMERAS 5 CELDAS DE CÓDIGO DEL NOTEBOOK DE REGULARIZACIÓN - IBM Machine Learning



Celda 1: Instalación de librerías necesarias

```
python

# Install the required libraries
!pip install -U scikit-learn
!pip install pandas
!pip install numpy
!pip install seaborn
!pip install matplotlib
```



Explicación:

Estas líneas instalan las bibliotecas necesarias para el proyecto:

- **scikit-learn**: Para modelos de machine learning.
- **pandas**: Manipulación y análisis de datos tabulares.
- **numpy**: Computación numérica con arrays.
- **seaborn** y **matplotlib**: Visualización de datos.

✅ Estas instrucciones aseguran que el entorno esté preparado para ejecutar correctamente el resto del notebook.



Celda 2: Supresión de advertencias

```
python

# Surpress warnings:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

Explicación:

Esta celda redefine la función `warnings.warn` con una versión vacía (`pass`), lo cual:

- Suprime todas las advertencias durante la ejecución del notebook.
- Es útil cuando se quiere tener una salida limpia, aunque se deben usar con precaución ya que puede ocultar errores importantes.

Celda 3: Importación de bibliotecas

```
python

import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Librerías generales:

- **pandas** y **numpy**: para manipulación de datos.
- **seaborn** y **matplotlib**: para visualización de gráficos.
- `%matplotlib inline`: directiva de Jupyter para mostrar gráficos embebidos.

Librerías de Scikit-learn importadas:

```
python

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures, scale, StandardScaler, MinMaxScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
```

Modelos de regresión:

- **LinearRegression**: Regresión lineal básica.
- **Ridge**: Regresión con regularización L2.
- **Lasso**: Regularización L1.
- **ElasticNet**: Combinación de L1 y L2.

Preprocesamiento:

- Escalado (`scale`, `StandardScaler`, `MinMaxScaler`).
- Polinomios (`PolynomialFeatures`).

Optimización y evaluación:

- `train_test_split`: división de datos en entrenamiento y prueba.
- `r2_score`: métrica de evaluación del modelo.
- `GridSearchCV`: búsqueda de hiperparámetros óptimos.
- `SelectKBest`, `f_regression`: selección de variables relevantes.
- `PCA`: reducción de dimensionalidad.

Celda 4: Visualización del R^2 para cada feature individual

python

```
def get_R2_features(model, test=True):
    # X: global
    features = list(X)
    features.remove("three")

    R_2_train = []
    R_2_test = []

    for feature in features:
        model.fit(X_train[[feature]], y_train)

        R_2_test.append(model.score(X_test[[feature]], y_test))
        R_2_train.append(model.score(X_train[[feature]], y_train))

    plt.bar(features, R_2_train, label="Train")
    plt.bar(features, R_2_test, label="Test")
    plt.xticks(rotation=90)
    plt.ylabel(" $R^2$ ")
    plt.legend()
    plt.show()

    print("Training  $R^2$  mean value {} Testing  $R^2$  mean value {}".format(
        str(np.mean(R_2_train)), str(np.mean(R_2_test))))
    print("Training  $R^2$  max value {} Testing  $R^2$  max value {}".format(
        str(np.max(R_2_train)), str(np.max(R_2_test))))
```

Funcionalidad:

- Analiza qué tan bien predice cada variable individualmente el target (`y_train`, `y_test`) usando un modelo simple.
- Muestra la métrica R^2 para cada feature en entrenamiento y prueba.
- Elimina la variable "three" del análisis (posiblemente por no ser útil).

Interpretación:

- Un R^2 cercano a 1 indica buena capacidad predictiva.
- Esta función ayuda a identificar variables que por sí solas explican bien la variable dependiente.

Celda 5: Visualización de los coeficientes del modelo

python

```
def plot_coef(X, model, name=None):
    plt.bar(X.columns[2:], abs(model.coef_[2:]))
    plt.xticks(rotation=90)
    plt.ylabel("$coefficients$")
    plt.title(name)
    plt.show()

print("R^2 on training data ", model.score(X_train, y_train))
print("R^2 on testing data ", model.score(X_test, y_test))
```

Funcionalidad:

- Grafica los coeficientes (valores absolutos) de un modelo de regresión entrenado.
- Omite las dos primeras columnas de x (probablemente categóricas o poco informativas).
- Imprime los R^2 para entrenamiento y test, lo que permite detectar overfitting o underfitting.

Importancia:

- Útil para visualizar qué variables tienen mayor peso en la predicción.
- Permite interpretar modelos lineales y regularizados (Ridge, Lasso, ElasticNet).

CONCLUSIÓN PARCIAL DEL FUNCIONAMIENTO (hasta Celda 5)

Este notebook está diseñado para:

- Entrenar y evaluar modelos de regresión con distintos tipos de regularización.
- Visualizar y analizar el impacto de cada variable sobre la variable objetivo.
- Medir la capacidad predictiva usando métricas como R^2 y coeficientes del modelo.

Ya se definieron funciones para evaluar el poder predictivo de variables individuales y para interpretar modelos lineales.

TEORÍA COMPLEMENTARIA SOBRE REGULARIZACIÓN

¿Qué es la regularización?

La regularización es una técnica para reducir el sobreajuste en modelos de regresión y clasificación al penalizar la magnitud de los coeficientes.

Tipos:

- **Ridge (L2):** Penaliza la suma de los cuadrados de los coeficientes. Minimiza:

$$\min \sum (y_i - \hat{y}_i)^2 + \alpha \sum \beta_j^2$$

- **Lasso (L1):** Penaliza la suma de los valores absolutos:

$$\min \sum (y_i - \hat{y}_i)^2 + \alpha \sum |\beta_j|$$

- **ElasticNet:** Combina ambas penalizaciones (L1 + L2).

Beneficios:

- Reduce varianza del modelo.
- Selecciona variables relevantes automáticamente (Lasso).
- Mejora generalización en conjunto de test.

¿Deseas que continúe con las siguientes celdas?



CONTINUA

EXPLICACIÓN DETALLADA DE LAS CELDAS 6 A 10

Celda 6: Visualización de valores reales vs valores predichos

python

```
def plot_dis(y, yhat):
    plt.figure()
    ax1 = sns.distplot(y, hist=False, color="r", label="Actual Value")
    sns.distplot(yhat, hist=False, color="b", label="Fitted Values", ax=ax1)
    plt.legend()

    plt.title('Actual vs Fitted Values')
    plt.xlabel('Price (in dollars)')
    plt.ylabel('Proportion of Cars')

    plt.show()
    plt.close()
```

Explicación:

- Esta función compara visualmente los valores reales (**y**) y los valores predichos (**yhat**) con gráficos de densidad (**distplot**).
- Ayuda a verificar si el modelo predice correctamente la distribución de la variable objetivo (en este caso, el **precio de autos**).

 **Nota técnica:** `sns.distplot` está **deprecated**. La alternativa moderna es `sns.kdeplot`.


Celda 7: Carga de datos

python

```
data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-ML240EN-SkillsNetwork/labs/encoded_car_data.csv')
data.head()
```

Explicación:

- Carga el dataset desde un URL público proporcionado por IBM.
- Muestra las primeras filas con `data.head()`.

 Este dataset está **preprocesado y codificado**, por lo tanto, probablemente ya tenga variables numéricas listas para usar en modelos de regresión.

Celda 8: Información general del dataset

```
python

data.info()
```

Explicación:

- Muestra estructura del DataFrame:
 - Nombres de columnas
 - Tipos de datos (`int64`, `float64`, etc.)
 - Valores nulos
 - Tamaño

✅ Permite confirmar que los datos están completos y en formato adecuado para entrenamiento.

Celda 9: Separación de variables predictoras y objetivo

```
python

X = data.drop('price', axis=1)
y = data.price
```

Explicación:

- `x`: contiene todas las columnas excepto `price`, que es la **variable a predecir**.
- `y`: es la columna `price`.

Esto separa el dataset en:

- **Variables independientes (predictoras)**: atributos del auto
- **Variable dependiente (target)**: precio

Celda 10: División en conjunto de entrenamiento y prueba

```
python

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
print("number of test samples :", X_test.shape[0])
```

```
print("number of training samples:", X_train.shape[0])
```

Explicación:

- `train_test_split`: divide el dataset en 90% entrenamiento y 10% prueba.
- `random_state=42`: garantiza reproducibilidad (misma división cada vez).
- Se imprimen las cantidades de muestras para cada conjunto.

📌 Esta división es fundamental para evaluar correctamente el rendimiento del modelo en datos **nunca vistos** (generalización).

✅ CONCLUSIÓN PARCIAL (Celdas 6 a 10)

Estas celdas completan la **fase de preparación del dataset**:

- Se carga un dataset de autos codificado.
- Se analiza su estructura y se separa en features y target.
- Se divide en conjuntos de entrenamiento y prueba.
- Además, se define una función de visualización clave para comparar predicción y realidad.

Esto sienta las bases para aplicar los modelos de regresión (con y sin regularización) en las próximas celdas.

TEORÍA COMPLEMENTARIA: EVALUACIÓN DE MODELOS DE REGRESIÓN

¿Por qué dividir en entrenamiento y prueba?

Para evitar sobreajuste (**overfitting**) y medir la capacidad real de generalización del modelo.

¿Qué mide el gráfico `plot_dis()`?

- Evalúa visualmente si el modelo capta correctamente la distribución de precios.
- Si las curvas de `y` y `yhat` se superponen → buen modelo.

¿Continuo con las próximas celdas del notebook?



CONTINUA

✅ EXPLICACIÓN DETALLADA DE LAS CELDAS 11 A 15

Celda 11: Creación del modelo de Regresión Lineal

```
python

lm = LinearRegression()
```

Explicación:

- Se crea una instancia del modelo **Regresión Lineal** de Scikit-learn.
- Este modelo busca ajustar una recta (o hiperplano en dimensiones superiores) que minimice el **error cuadrático medio** entre las predicciones y los valores reales.

Fórmula del modelo:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

Donde:

- \hat{y} : valor predicho
- β_0 : intercepto
- β_i : coeficientes o pesos

Celda 12: Entrenamiento del modelo

```
python

lm.fit(X_train, y_train)
```

Explicación:

- El modelo aprende los coeficientes β_i que mejor ajustan los datos de entrenamiento usando **mínimos cuadrados ordinarios (OLS)**.
- Internamente, busca minimizar la siguiente función de pérdida:

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

Celda 13: Predicción en datos de test

```
python

predicted = lm.predict(X_test)
```

Explicación:

- Se utilizan los valores de **x_test** para predecir los precios (**predicted**) usando los coeficientes aprendidos en el entrenamiento.
- Esta es una evaluación “fuera de muestra”, que refleja la capacidad del modelo para generalizar.

Celda 14: Evaluación con métrica R^2

python

```
print("R^2 on training data ", lm.score(X_train, y_train))
print("R^2 on testing data ", lm.score(X_test, y_test))
```

Explicación:

- Se calcula el **coeficiente de determinación** R^2 para los conjuntos de entrenamiento y prueba.
- R^2 mide la proporción de varianza explicada por el modelo:

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

 Valores típicos:

- $R^2 \approx 1$: excelente ajuste
- $R^2 \approx 0$: mal ajuste
- $R^2 < 0$: el modelo es peor que simplemente predecir el promedio

Comparar el R^2 de entrenamiento vs prueba nos ayuda a detectar **overfitting**.

Celda 15: Visualización de la predicción vs datos reales

python

```
plot_dis(y_test, predicted)
```

Explicación:

- Se llama a la función definida previamente para visualizar la comparación entre los precios reales (**y_test**) y los predichos (**predicted**).
- Dos curvas similares indican que el modelo está aprendiendo correctamente.

CONCLUSIÓN PARCIAL (Celdas 11 a 15)

Estas celdas implementan y evalúan el **modelo base de regresión lineal**:

1. Se crea el modelo con **LinearRegression**.
2. Se entrena con **X_train, y_train**.
3. Se predicen precios con **X_test**.
4. Se evalúa el rendimiento con R^2 y una visualización de densidad.

 Sirve como **modelo de referencia** (baseline) antes de introducir regularización.

TEORÍA ADICIONAL: Regresión Lineal

Supuestos del modelo lineal clásico:

1. **Linealidad** entre variables independientes y dependiente.
2. **Independencia** de los errores.
3. **Homoscedasticidad**: varianza constante de los errores.
4. **No multicolinealidad**.
5. **Normalidad** de los residuos.

Problemas comunes:

- **Sobreajuste (overfitting)**: muchos coeficientes grandes → modelo demasiado flexible.
- **Multicolinealidad**: cuando las variables predictoras están correlacionadas entre sí, puede afectar la estabilidad de los coeficientes.

→ **La regularización (Ridge, Lasso, ElasticNet)** se introduce justamente para solucionar estos problemas.

¿Deseas que continúe con las próximas celdas del notebook?



CONTINUA

EXPLICACIÓN DETALLADA DE LAS CELDAS 16 A 20

Celda 16: Visualización de coeficientes del modelo lineal

```
python

plot_coef(X, lm, name="Linear Regression")
```

Explicación:

- Se grafica la magnitud absoluta de los coeficientes del modelo `lm` (regresión lineal).
- Esto permite ver qué variables tienen mayor influencia sobre la predicción del precio.

Utilidad práctica:

- Coeficientes grandes pueden indicar variables dominantes, pero también pueden revelar **multicolinealidad** o **sobreajuste**.
- Es un paso clave antes de aplicar regularización, ya que **Ridge y Lasso penalizan justamente estos coeficientes grandes**.

Celda 17: Definición del modelo Ridge (Regularización L2)

```
python


rr = Ridge(alpha=0.01)
rr
```

Explicación:

- Se crea una instancia del modelo **Ridge** (regresión lineal con penalización L2).
- El parámetro **alpha=0.01** controla la fuerza de la penalización.
 - Si **alpha** \rightarrow 0, se parece a una regresión lineal común.
 - Si **alpha** es grande, se penalizan fuertemente los coeficientes.

Función de costo Ridge:

$$\text{Loss} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^p \beta_j^2$$

 Esto ayuda a controlar:

- Sobreajuste
- Multicolinealidad

Celda 18: Entrenamiento del modelo Ridge

```
python

rr.fit(X_train, y_train)
```

Explicación:

- El modelo ajusta sus coeficientes a los datos de entrenamiento utilizando la penalización L2.
- El objetivo es encontrar un balance entre ajuste y complejidad del modelo.

Celda 19: Predicción con Ridge

```
python

rr.predict(X_test)
```

Explicación:

- El modelo Ridge genera predicciones para los datos de prueba, igual que antes.
- En la siguiente celda se evalúan los resultados.

Celda 20: Evaluación del modelo Ridge con R^2

python

```
print("R^2 on training data ", rr.score(X_train, y_train))
print("R^2 on testing data ", rr.score(X_test, y_test))
```

Explicación:

- Se calcula el R^2 para Ridge en datos de entrenamiento y test.
- Comparando con el modelo lineal:
 - Si mejora el R^2 en test y baja levemente en entrenamiento → mejor generalización.
 - Si ambos bajan, el modelo está subajustando (underfitting) → aumentar α puede no ser buena idea.

CONCLUSIÓN PARCIAL (Celdas 16 a 20)

Se comienza la comparación entre **regresión lineal sin regularización** y **regresión Ridge**:

- Se observa que Ridge reduce posibles problemas de sobreajuste.
- Se entrena un nuevo modelo con penalización L2 para controlar la magnitud de los coeficientes.
- Se mantiene la métrica de evaluación R^2 para comparar con el modelo lineal base.

Este paso permite visualizar cómo los coeficientes cambian y si la **capacidad de generalización mejora**.

TEORÍA COMPLEMENTARIA: RIDGE REGRESSION

¿Qué hace Ridge?

- Penaliza los coeficientes grandes que podrían indicar sobreajuste.
- A diferencia de Lasso, **no fuerza coeficientes a cero**, por lo que **no realiza selección de variables**.
- Es útil cuando:
 - Hay muchas variables correlacionadas.
 - Hay más variables que observaciones (caso de alta dimensionalidad).

¿Deseas que continúe con las siguientes celdas donde probablemente se explore Lasso y ElasticNet?



CONTINUA

EXPLICACIÓN DETALLADA DE LAS CELDAS 21 A 25

Celda 21: Comparación visual entre Regresión Lineal y Ridge

```
python

plot_coef(X, lm, name="Linear Regression")
plot_coef(X, rr, name="Ridge Regression")
```

Explicación:

- Se grafican los coeficientes del modelo **sin regularización (lm)** y del modelo **Ridge (rr)**.
- Esto permite visualizar cómo **Ridge reduce** los coeficientes en comparación con la regresión lineal clásica.

Conclusión esperada:

- Ridge tiende a reducir la magnitud de los coeficientes, distribuyéndolos más uniformemente.
- Previene que ciertas variables dominen excesivamente el modelo.

Celda 22: Ridge con penalización más fuerte (alpha = 1)

```
python

rr = Ridge(alpha=1)
rr.fit(X_train, y_train)
plot_coef(X, rr)
```

Explicación:

- Se reentrena el modelo Ridge con **alpha=1**, una penalización más fuerte.
- Se grafica nuevamente para observar cómo cambian los coeficientes.

A medida que **alpha** aumenta:

- La penalización es más agresiva.
- Los coeficientes se encogen más hacia cero.
- Puede aumentar el sesgo, pero reduce varianza → mejora generalización.

Celda 23: Análisis del efecto de alpha sobre coeficientes y R^2

```
python

alphas = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
R_2 = []
coefs = []

for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    coefs.append(abs(ridge.coef_))
    R_2.append(ridge.score(X_test, y_test))
```

+ Gráfico 1: Regularization Path

```
python

ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale("log")
plt.xlabel("alpha")
plt.ylabel("weights")
plt.title("Ridge coefficients as a function of the regularization (regularization path)")
plt.show()
```

- Muestra cómo los coeficientes disminuyen al aumentar **alpha**.
- Eje x en escala logarítmica.

Gráfico 2: R^2 vs α

```
python

ax = plt.gca()
ax.plot(alphas, R_2)
ax.set_xscale("log")
plt.xlabel("alpha")
plt.ylabel("$R^2$")
plt.title("$R^2$ as a function of the regularization")
plt.show()
```

- Permite encontrar el **alpha** que maximiza el R^2 en el set de prueba.
- Visualiza el **trade-off entre sesgo y varianza**.

Celda 24: Error cuadrático medio (MSE) vs alpha

```
python

alphas = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
MEAN_SQE = []

for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    MEAN_SQE.append(mean_squared_error(ridge.predict(X_test), y_test))
```

Gráfico:

```
python

ax = plt.gca()
ax.plot(alphas, MEAN_SQE)
ax.set_xscale("log")
plt.xlabel("alpha")
plt.ylabel("MSE")
plt.title("$MSE$ as a function of the regularization")
plt.show()
```

Interpretación:

- Se observa cómo varía el error cuadrático medio según la penalización.
- El punto mínimo indica el **alpha** óptimo para minimizar el error de predicción.

Celda 25: Pipeline con Ridge, estandarización y polinomios

```
python

Input = [
    ('polynomial', PolynomialFeatures(include_bias=False, degree=2)),
    ('ss', StandardScaler()),
    ('model', Ridge(alpha=1))
]
pipe = Pipeline(Input)
```

Explicación:

- Se construye un **Pipeline** con 3 etapas:
 1. **PolynomialFeatures**: genera variables polinomiales hasta grado 2.
 2. **StandardScaler**: normaliza todas las variables.
 3. **Ridge**: aplica el modelo regularizado.

Ventajas del Pipeline:

- Encapsula todos los pasos del preprocesamiento y modelado en una sola estructura.
- Evita fugas de datos (**data leakage**).
- Facilita búsqueda de hiperparámetros con **GridSearchCV**.

CONCLUSIÓN PARCIAL (Celdas 21 a 25)

Estas celdas profundizan el análisis de **Ridge Regression**:

1. Se visualiza el efecto de distintos valores de **alpha** sobre:
 - Los coeficientes.
 - El R^2 de test.
 - El error cuadrático medio (MSE).
2. Se construye un **Pipeline profesional** con polinomios + normalización + modelo.

Este análisis permite:

- Elegir el **alpha** óptimo.
- Controlar el trade-off entre **ajuste y generalización**.
- Prever cómo la complejidad del modelo afecta su rendimiento.

TEORÍA COMPLEMENTARIA: REGULARIZATION PATH

- Un **regularization path** muestra cómo cambian los coeficientes al variar **alpha**.
- Este tipo de gráfico ayuda a:
 - Seleccionar hiperparámetros.
 - Detectar variables que permanecen relevantes incluso con regularización fuerte.

¿Deseas que continúe con las siguientes celdas? Posiblemente ahora venga Lasso o ElasticNet.



CONTINUA

✓ EXPLICACIÓN DETALLADA DE LAS CELDAS 26 A 30


Celda 26: Entrenamiento del Pipeline

```
python

pipe.fit(X_train, y_train)
```

Explicación:

- Se entrena el **Pipeline** definido en la celda anterior con:
 - **Features polinomiales** de grado 2 (**PolynomialFeatures**)
 - **Normalización** (**StandardScaler**)
 - **Modelo Ridge** con **alpha = 1**

 Esta estructura es robusta para manejar modelos no lineales con regularización, asegurando el correcto orden de los pasos.


Celda 27: Predicción y evaluación del Pipeline

```
python

predicted = pipe.predict(X_test)
pipe.score(X_test, y_test)
```

Explicación:

- Se generan predicciones sobre **X_test** con el pipeline entrenado.
- **pipe.score(...)** devuelve el R^2 del modelo Ridge regularizado aplicado a features polinomiales.

 Aquí ya se están utilizando **regresores no lineales regularizados**, lo que puede mejorar la capacidad predictiva del modelo respecto al lineal clásico.

Celda 28: Definición de la grilla de hiperparámetros

```
python

param_grid = {
    "polynomial__degree": [1, 2, 3, 4],
    "model__alpha": [0.0001, 0.001, 0.01, 0.1, 1, 10]
}
```

Explicación:

- Se define una grilla de búsqueda para optimizar:
 - El grado del polinomio (1° a 4°)
 - El parámetro de regularización **alpha** del modelo Ridge

 Nótese que:

- "**polynomial__degree**" se refiere al nombre del paso en el pipeline (**polynomial**) y su parámetro interno (**degree**).
- "**model__alpha**" corresponde a **alpha** dentro de **model** (Ridge).


Celda 29: Búsqueda de hiperparámetros con GridSearchCV

```
python

search = GridSearchCV(pipe, param_grid, n_jobs=2)
search.fit(X_train, y_train)
search
```

Explicación:

- Se utiliza **validación cruzada** para encontrar la combinación óptima de **degree** y **alpha** que maximice el rendimiento del modelo.
- **GridSearchCV** prueba cada combinación de la grilla usando varias particiones de entrenamiento y validación.
- **n_jobs=2**: utiliza 2 núcleos de CPU para acelerar el proceso.

 Esta es una forma robusta de optimizar modelos sin sesgo por una sola partición de datos.

Celda 30: Visualización de los resultados de la búsqueda

```
python

pd.DataFrame(search.cv_results_).head()
```

Explicación:

- Se convierte el resultado de **GridSearchCV** a un **DataFrame** para inspeccionar:
 - Cada combinación de hiperparámetros

- El rendimiento promedio en validación (`mean_test_score`)
- El desvío estándar de cada validación
- Permite tomar decisiones informadas sobre qué configuración usar.

✓ CONCLUSIÓN PARCIAL (Celdas 26 a 30)

En estas celdas se lleva a cabo una optimización **profesional y sistemática** del modelo Ridge con características polinomiales. En resumen:

- Se entrena un pipeline completo.
- Se evalúa su desempeño con R^2 .
- Se realiza una búsqueda en grilla para encontrar:
 - El mejor grado de polinomio.
 - El mejor **alpha** para regularización.

Este enfoque permite balancear **sesgo-varianza** de forma automática y encontrar modelos óptimos en contextos no lineales.

📖 TEORÍA COMPLEMENTARIA: PIPELINES Y GRID SEARCH

🔄 Pipelines

- Permiten encadenar múltiples pasos de transformación y modelado.
- Ayudan a evitar **data leakage** y facilitan la validación cruzada conjunta.

🔍 GridSearchCV

- Prueba todas las combinaciones posibles de parámetros definidos.
- Usa **validación cruzada** interna para evaluar cada combinación.
- Escoge el modelo con mejor puntaje promedio.

¿Querés que continúe con las siguientes celdas del notebook? Posiblemente aparezcan ahora modelos Lasso o ElasticNet.



CONTINUA

✓ EXPLICACIÓN DETALLADA DE LAS CELDAS 31 A 35

🏆 Celda 31: Resultados óptimos de Grid Search

```
python

print("best_score_: ", search.best_score_)
print("best_params_: ", search.best_params_)
```

Explicación:

- **search.best_score_:** muestra el mejor valor promedio de R^2 obtenido durante la validación cruzada.
- **search.best_params_:** indica la combinación óptima de parámetros (**degree**, **alpha**) que generó ese resultado.

 Esto es fundamental para:

- Verificar la mejora con respecto al modelo base.
- Saber qué configuración utilizar en producción.


Celda 32: Predicción usando el mejor modelo encontrado

```
python

predict = search.predict(X_test)
predict
```

Explicación:

- Se utiliza directamente el objeto **search** para predecir con el modelo entrenado con los mejores hiperparámetros.
- Las predicciones se almacenan en **predict**.

 Esto demuestra que **GridSearchCV** mantiene internamente el mejor modelo (**best_estimator_**).


Celda 33: Extraer el mejor modelo estimado

```
python

best = search.best_estimator_
best
```

Explicación:

- Se accede al modelo "ganador" del **GridSearchCV** y se guarda en **best**.
- Este modelo es un pipeline con:
 - El mejor grado de polinomio
 - La mejor regularización **alpha**

 Esto permite usar el modelo sin necesidad de reentrenarlo.

Celda 34: Predicción con best

```
python

predict = best.predict(X_test)
predict
```

Explicación:

- Confirmación de que el modelo extraído (**best**) produce las mismas predicciones que `search.predict(...)`.

Celda 35: Evaluación del mejor modelo en test

```
python

best.score(X_test, y_test)
```

Explicación:

- Se calcula el R^2 final del modelo óptimo sobre los datos de prueba.
- Este valor es la **evaluación definitiva** de qué tan bien generaliza el modelo después de todo el proceso de optimización.

CONCLUSIÓN PARCIAL (Celdas 31 a 35)

Estas celdas completan el ciclo de entrenamiento y evaluación del modelo Ridge optimizado:

- Se identifican y aplican los **mejores hiperparámetros**.
- Se evalúa su capacidad predictiva sobre el conjunto de test.
- Se valida que el pipeline funciona de forma coherente y automática.

 Este enfoque profesional garantiza:

- La mejor generalización posible bajo el marco de Ridge.
- Un modelo final listo para usar en producción o para compararse con otros enfoques como Lasso o ElasticNet.

TEORÍA COMPLEMENTARIA: BEST ESTIMATOR Y VALIDACIÓN

`best_estimator_`:

- Ahorra tiempo y recursos.
- Garantiza que el modelo final tenga los parámetros óptimos, ya reentrenado internamente con los datos de entrenamiento.

Evaluación en test:

- Debe realizarse solo una vez, al final del pipeline.
- Es la mejor estimación de rendimiento futuro del modelo.

¿Deseas que continúe con las siguientes celdas? Si hay más contenido, probablemente aparezcan ahora otros modelos de regularización como **Lasso** o **ElasticNet**.



CONTINUA


EXPLICACIÓN DETALLADA DE LAS CELDAS 36 A 40

Celda 36: Reentrenar el mejor modelo con todos los datos

```
python  
  
best.fit(X, y)
```

Explicación:

- Una vez encontrado el mejor modelo mediante validación cruzada, se entrena nuevamente con **todo el conjunto de datos** disponibles (X, y).
- Esto mejora el poder predictivo del modelo para futuras predicciones, ya que se aprovechan más datos sin usar test en esta etapa.

 Este paso es válido **solo después** de haber hecho la evaluación final en test.

Celda 37: Selección de columnas para análisis gráfico

```
python  
  
columns = ['wheelbase', 'curbweight', 'enginesize', 'bore', 'horsepower',  
           'carlength', 'carwidth', 'citympg']
```

Explicación:

- Se selecciona un subconjunto de variables predictoras relevantes del dataset.
- Estas variables se usarán para visualizar la relación entre **una sola feature y el precio**, ajustando modelos univariados.

Celda 38: Visualización de relación entre variables y predicción

python

```
for column in columns:
    search.fit(X_train[[column]], y_train)
    x = np.linspace(X_test[[column]].min(), X_test[[column]].max(), num=100)
    plt.plot(x, search.predict(x.reshape(-1, 1)), label="prediction")
    plt.plot(X_test[column], y_test, 'ro', label="y")
    plt.xlabel(column)
    plt.ylabel("y")
    plt.legend()
    plt.show()
```

Explicación:

- Se entrena el pipeline **search** (con polinomios y Ridge) para **cada variable individualmente**.
- Se grafica:
 - La predicción (línea azul).
 - Los valores reales (puntos rojos).

Este enfoque permite:

- Visualizar la forma funcional (lineal, cuadrática, etc.) entre una sola variable y el target.
- Analizar la capacidad del modelo para capturar esa relación.

Celda 39: Entrenamiento del modelo Lasso (Regularización L1)

python

```
la = Lasso(alpha=0.1)
la.fit(X_train, y_train)
la
```

Explicación:

- Se entrena un modelo **Lasso** (Least Absolute Shrinkage and Selection Operator) con **alpha = 0.1**.
- A diferencia de Ridge, Lasso **puede llevar coeficientes exactamente a cero**, realizando **selección automática de variables**.

Función de pérdida:

$$\text{Loss} = \sum (y_i - \hat{y}_i)^2 + \alpha \sum |\beta_j|$$

Lasso es muy útil cuando:

- Hay muchas variables.
- Se desea identificar cuáles son realmente importantes.

Celda 40: Predicción con Lasso

```
python
```

```
predicted = la.predict(X_test)
predicted
```

Explicación:

- Se generan predicciones con el modelo Lasso entrenado anteriormente.
- Estas predicciones luego pueden compararse visual o cuantitativamente con las del modelo Ridge o Lineal.

CONCLUSIÓN PARCIAL (Celdas 36 a 40)

En esta sección se combinan técnicas de visualización e introducción a nuevos modelos:

1. Se reentrena el mejor modelo para aprovechar todos los datos.
2. Se visualiza la relación entre variables individuales y el precio.
3. Se introduce el modelo **Lasso** como una alternativa regularizada:
 - Penaliza con L1.
 - Permite **sparse solutions** (coeficientes exactamente cero).
 - Es ideal para selección de variables.

TEORÍA COMPLEMENTARIA: LASSO REGRESSION

Características principales:

- Reduce la complejidad del modelo.
- Puede eliminar por completo variables irrelevantes.
- Más útil que Ridge cuando el número de variables es alto y se busca interpretabilidad.

Diferencias clave con Ridge:

Aspecto	Ridge (L2)	Lasso (L1)
Penalización	$\sum \beta^2$	\sum
Coefs = 0	No	Sí
Selección vars	No	Sí
Multicolineal.	Bueno	Menos eficaz

¿Deseás que continúe con las últimas celdas? Puedo avisarte también cuando no queden más bloques por explicar.



CONTINUA

✓ EXPLICACIÓN DETALLADA DE LAS CELDAS 41 A 45

Celda 41: Evaluación de regresión lineal nuevamente

python

```
print("R^2 on training data ", lm.score(X_train, y_train))
print("R^2 on testing data ", lm.score(X_test, y_test))
```

Explicación:

- Se vuelve a imprimir el rendimiento del modelo de regresión lineal (`lm`) sobre los conjuntos de entrenamiento y prueba.
- Probablemente para compararlo directamente con los modelos Ridge y Lasso entrenados recientemente.

 Comparar los tres modelos ayuda a elegir el mejor balance entre complejidad y generalización.

Celda 42: Comparación visual entre Ridge y Lasso

python

```
plot_coef(X, rr, name="Ridge Regression")
plot_coef(X, la, name="Lasso Regression")
```

Explicación:

- Se grafican los coeficientes de ambos modelos regularizados:
 - `rr`: Ridge
 - `la`: Lasso

Expectativas:

- **Ridge** reduce todos los coeficientes, pero no los anula.
- **Lasso** puede llevar coeficientes a cero → selección automática de variables.

Celda 43: Regularization Path de Lasso

python

```
alphas = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
R_2 = []
coefs = []

for alpha in alphas:
    la = Lasso(alpha=alpha)
    la.fit(X_train, y_train)
```



```
coefs.append(abs(la.coef_))
R_2.append(la.score(X_test, y_test))
```

+ Gráfico 1: Coeficientes vs Alpha

```
python

ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale("log")
plt.xlabel("alpha")
plt.ylabel("weights")
plt.title("Ridge coefficients as a function of the regularization (regularization path)")
plt.show()
```

⚠ El título debería decir "Lasso", no "Ridge", pero el análisis es válido.

Gráfico 2: R^2 vs Alpha

```
python

ax = plt.gca()
ax.plot(alphas, R_2)
ax.set_xscale("log")
plt.xlabel("alpha")
plt.ylabel("$R^2$")
plt.title("$R^2$ as a function of the regularization")
plt.show()
```

Interpretación:

- A medida que **alpha** aumenta:
 - Los coeficientes se reducen y algunos se anulan (coef = 0).
 - El R^2 puede mejorar hasta cierto punto, luego disminuir por underfitting.

Celda 44: Pipeline con Lasso y variables polinomiales

```
python

Input = [
    ('polynomial', PolynomialFeatures(include_bias=False, degree=2)),
    ('ss', StandardScaler()),
    ('model', Lasso(alpha=1, tol=0.2))
]
pipe = Pipeline(Input)
```

Explicación:

- Se define un pipeline similar al usado con Ridge, pero ahora con **Lasso**.
- **tol=0.2**: tolerancia en la optimización; un valor más alto puede acelerar la convergencia pero disminuir precisión.

🔪 Este pipeline permite trabajar con relaciones no lineales y realizar selección automática de variables.

Celda 45: Entrenamiento y predicción con Pipeline Lasso

```
python

pipe.fit(X_train, y_train)
pipe.predict(X_test)
```

Explicación:


- Se entrena el pipeline sobre los datos de entrenamiento.
- Se generan predicciones con el modelo regularizado usando Lasso.

 Permite comparar con los resultados obtenidos con Ridge y con el modelo base lineal.

CONCLUSIÓN PARCIAL (Celdas 41 a 45)

Estas celdas completan la introducción y análisis del modelo **Lasso Regression**:

- Se comparan visualmente los coeficientes generados por Ridge y Lasso.
- Se explora cómo el parámetro **alpha** afecta los coeficientes y el rendimiento del modelo.
- Se construye un pipeline con Lasso y se entrena sobre features polinomiales.

 Lasso es especialmente útil en problemas de alta dimensionalidad o cuando se busca interpretar qué variables son verdaderamente relevantes.

TEORÍA COMPLEMENTARIA: CUANDO USAR LASSO

- Cuando el dataset tiene muchas variables y se quiere reducir dimensionalidad automáticamente.
- Ideal si hay **sospecha de irrelevancia** en muchas features.
- Mejora la **interpretabilidad** del modelo.

¿Querés que continúe con las siguientes celdas y te avise cuando se haya terminado todo el código del notebook?



CONTINUA

EXPLICACIÓN DETALLADA DE LAS CELDAS 46 A 50

Celda 46: Evaluación del pipeline con Lasso

```
python

print("R^2 on training data ", pipe.score(X_train, y_train))
```

```
print("R^2 on testing data ", pipe.score(X_test, y_test))
```

Explicación:

- Evalúa el rendimiento del pipeline que incluye:
 - Expansión polinomial
 - Normalización
 - Modelo Lasso con **alpha=1**, **tol=0.2**
- El score corresponde a R^2 sobre entrenamiento y prueba.

 Esta comparación ayuda a detectar si el modelo:

- Tiene buen ajuste (R^2 alto)
- Sufre de overfitting (alta varianza entre train/test)
- O de underfitting (ambos R^2 bajos)


Celda 47: Definición de grilla de hiperparámetros para Lasso

```
python

param_grid = {
    "polynomial_degree": [1, 2, 3, 4, 5],
    "model__alpha": [0.0001, 0.001, 0.01, 0.1, 1, 10]
}
```

Explicación:

- Se establece una grilla para buscar:
 - Grado del polinomio: desde lineal hasta quíntico.
 - Parámetro **alpha** de regularización para Lasso.

 Al combinar ambos, se puede:

- Capturar relaciones más complejas entre variables.
- Controlar la complejidad y la cantidad de features activas.

Celda 48: Optimización con GridSearchCV

```
python

search = GridSearchCV(pipe, param_grid, n_jobs=1)
search.fit(X_train, y_train)
```

Explicación:

- Se aplica búsqueda en grilla con validación cruzada sobre el pipeline con Lasso.
- **n_jobs=1**: se usa un solo núcleo de CPU.

- Internamente, prueba todas las combinaciones de la grilla y calcula el R^2 medio en validación.

🔴 Se busca encontrar el modelo con mejor rendimiento general sin overfitting.

Celda 49: Extraer el mejor modelo ajustado

```
python

best = search.best_estimator_
best
```

Explicación:

- Se obtiene el mejor pipeline de la búsqueda.
- Este modelo ya está entrenado y listo para evaluar o usar en producción.

Celda 50: Evaluación final en test del mejor modelo

```
python

best.score(X_test, y_test)
```

Explicación:

- Se calcula el R^2 del mejor modelo sobre los datos de prueba.
- Esta es la métrica final que permite comparar los modelos Ridge vs Lasso bajo las mejores condiciones encontradas.

CONCLUSIÓN PARCIAL (Celdas 46 a 50)

En estas celdas se completa el proceso de ajuste de hiperparámetros para **Lasso**:

1. Se evalúa inicialmente un modelo con **alpha=1, degree=2**.
2. Se explora sistemáticamente diferentes combinaciones de **alpha** y **degree**.
3. Se entrena, evalúa y selecciona automáticamente el mejor modelo.
4. Se obtiene el rendimiento final en datos de prueba.

🔴 Este proceso es **análogo al usado con Ridge**, permitiendo una comparación justa entre métodos.

TEORÍA COMPLEMENTARIA: LASSO CON POLINOMIOS

- El uso de **PolynomialFeatures** puede generar muchas columnas adicionales.
- Lasso es útil aquí porque elimina automáticamente aquellas que **no aportan poder predictivo**.
- Combinado con **StandardScaler**, se evita que features con diferentes escalas dominen el modelo.

🔔 AVISO IMPORTANTE: YA NO QUEDAN MÁS CELDAS DE CÓDIGO EN EL NOTEBOOK.

A continuación, te daré una **conclusión global** de todo el notebook, junto con teoría adicional de valor. ¿Procedo?



CONTINUA

✅ CONCLUSIÓN GLOBAL DEL NOTEBOOK: Regularización - IBM Machine Learning

Este notebook constituye una guía práctica y sólida para aplicar técnicas de regresión regularizada (Ridge y Lasso) a un problema real: **predicción del precio de autos**. La estructura del análisis abarca todas las etapas necesarias en un flujo profesional de Machine Learning:

🕒 Resumen del Proceso Completo

1. Carga y exploración de datos:

- Se utiliza un dataset de autos codificado y limpio.
- Se separan features (x) y target (y).

2. Modelo base (LinearRegression):

- Se entrena un modelo sin regularización.
- Se evalúa usando R^2 y visualizaciones.

3. Aplicación de Ridge Regression (L2):

- Se ajusta con distintos valores de α .
- Se visualiza el efecto sobre los coeficientes y sobre R^2 .
- Se crea un pipeline completo con Ridge + features polinomiales.
- Se optimizan hiperparámetros con `GridSearchCV`.

4. Aplicación de Lasso Regression (L1):

- Se compara visualmente con Ridge.
- Se analiza cómo Lasso elimina coeficientes innecesarios.
- Se usa también con un pipeline y búsqueda de hiperparámetros.

5. Evaluación final:

- Se compara el rendimiento de todos los modelos entrenados.
- Se elige el modelo final según su R^2 sobre el conjunto de test.

📊 PRINCIPALES MÉTRICAS USADAS

- R^2 : coeficiente de determinación. Mide proporción de varianza explicada.
- **MSE (Error Cuadrático Medio)**: usado para comparar impacto del α en Ridge.

- **Coeficientes:** observados para estudiar regularización y relevancia de features.

HERRAMIENTAS CLAVE UTILIZADAS

- **scikit-learn:** modelado, pipelines, preprocesamiento, validación.
- **PolynomialFeatures, StandardScaler:** ingeniería de features y normalización.
- **Pipeline, GridSearchCV:** construcción y optimización modular del flujo.
- **Seaborn, Matplotlib:** visualización de coeficientes, predicciones y caminos de regularización.

TEORÍA Y CONCEPTOS CLAVE RESUMIDOS

1. Regresión Lineal

- Ajusta una línea o plano a los datos.
- Puede sobreajustarse si hay muchas variables irrelevantes o colineales.

2. Regularización

Controla la complejidad del modelo para mejorar la generalización:

Método	Penalización	Características
Ridge	$\sum \beta^2$	Reduce todos los coeficientes (shrinkage)
Lasso	$\sum \beta $	λ
ElasticNet	Combina ambas	Ideal para alta dimensionalidad + colinealidad

3. Pipeline

- Organiza pasos de procesamiento y modelado.
- Evita fugas de datos y facilita la validación.

4. Validación cruzada con GridSearchCV

- Evalúa múltiples combinaciones de hiperparámetros.
- Selecciona el mejor modelo en base a rendimiento promedio.

VENTAJAS DE ESTE ENFOQUE

- Modular, escalable, reutilizable.
- Visualmente explicativo (para stakeholders no técnicos).
- Profesional y alineado con estándares de la industria de ciencia de datos.

¿Qué podrías hacer a continuación?

1. **Agregar ElasticNet** para evaluar una combinación de L1 y L2.
 2. **Incluir más métricas** como MAE o RMSE.
 3. **Probar otros modelos:** regresores de árbol, SVM, redes neuronales.
 4. **Interpretación con SHAP o Permutation Importance.**
 5. **Evaluar estabilidad de coeficientes con bootstrap.**
-

¿Te gustaría que te ayude a extender este proyecto con ElasticNet o con visualizaciones adicionales comparativas? También puedo ayudarte a convertirlo en un **proyecto de portafolio profesional**.