Contenido

1. Regresión

La **regresión** es una técnica de análisis estadístico y aprendizaje automático utilizada para modelar la **relación entre una variable dependiente (respuesta)** y **una o más variables independientes (predictoras)**. Su propósito principal es predecir valores continuos de la variable dependiente en función de los valores de las variables independientes. A diferencia de la clasificación, que predice categorías, la regresión predice valores continuos, lo que la hace ideal para problemas como:

- Predicción de precios (casas, acciones, productos).
- Estimaciones de rendimiento (deportivos, financieros, académicos).
- Modelado de relaciones cuantitativas (crecimiento de población, evolución de ventas).

¿Cuándo Utilizar Regresión?

La regresión es adecuada cuando:

- El objetivo es predecir un valor continuo: Por ejemplo, estimar la temperatura o las ganancias.
- Existe una relación entre las variables: Especialmente si las variables independientes afectan de forma medible al resultado.
- Se necesita interpretar los resultados: Modelos simples como la regresión lineal permiten entender cómo cambian las predicciones en función de los cambios en las variables.

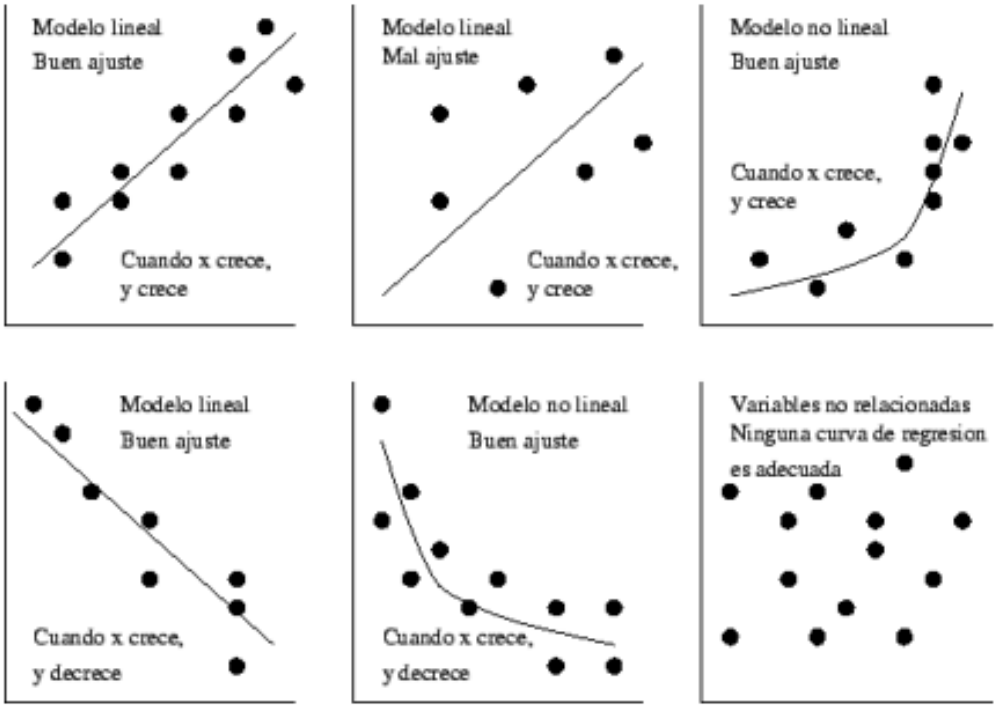
Conceptos Clave en Regresión

- **Función de Pérdida**: Durante el entrenamiento, el modelo minimiza una función de pérdida, como el **error cuadrático medio (MSE)**, que mide la diferencia entre los valores predichos y los valores reales.
- **Coeficientes o Parámetros**: Los coeficientes de cada variable independiente representan su peso o importancia en la predicción de la variable dependiente.
- **Regularización**: Técnica que agrega una penalización a los coeficientes para evitar el sobreajuste y mejorar la capacidad de generalización del modelo.

2. Tipos de Regresión

Existen diversos métodos de regresión, cada uno adecuado para diferentes tipos de problemas:

- **Regresión Lineal Simple**
- **Regresión Lineal Múltiple**
- **Regresión Polinomial**
- **Regresión Logística**



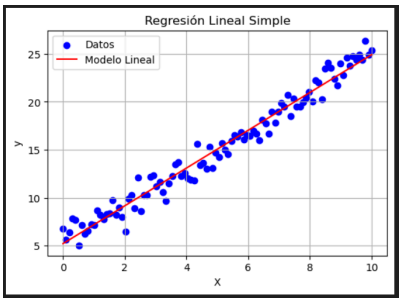
2.1. Regresión Lineal

2.1.1. Regresión Lineal Simple

- **Propósito**: Modelar una **relación lineal** entre una variable dependiente y y una sola variable independiente x. Se representa con una línea recta
- **Ecuación General**:

$$\hat{y} = \theta_0 + \theta_1 x_1$$

- **Características**:
  - Es el modelo más sencillo de regresión, con solo una variable independiente.
  - La relación entre x e y es **lineal** (representada por una línea recta).
  - Solo tiene un coeficiente (θ_1), que representa la pendiente de la línea.
- **Ejemplo**: Predecir el precio de un producto en función de una sola característica, como la cantidad de unidades vendidas. La relación se espera que sea una línea recta entre las dos variables.
- **Clase LinearRegression**: La clase de LinearRegression de Scikit-Learn tienen un coste computacional de $O(n^2)$. Si duplicamos el número de características, multiplicamos el tiempo de computación por 4, aproximadamente.
- **Visualización**:



- **Ejemplo código Python:**

```
import       as
from                    import
from                       import
```

```
from                import

# Datos de ejemplo
              1    2    3    4    5     # Variable independiente (e.g.,
unidades vendidas)
            2.5  4.5  6.5  8.5  10.5    # Variable dependiente (e.g.,
precios)

# Dividir los datos en conjunto de entrenamiento y prueba
                                                                    0.2
            42

# Crear el modelo de regresión lineal


# Entrenar el modelo


# Realizar predicciones


# Evaluar el modelo



print  Coeficiente  Pendiente                0
print  Intercepto
print  Error Cuadrático Medio  MSE
print  Coeficiente de Determinación

# Predicción para un nuevo dato
                       6

print f"Predicción para             0  0   unidades vendidas:",
          0

#Resultados
Coeficiente (Pendiente): 1.9999999999999996
Intercepto: 0.5000000000000018
Error Cuadrático Medio (MSE): 7.888609052210118e-31
Coeficiente de Determinación (R²): nan
Predicción para 6 unidades vendidas: 12.499999999999998
```

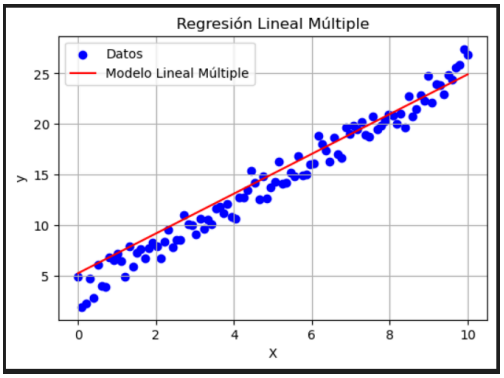
2.1.2. Regresión Lineal Múltiple

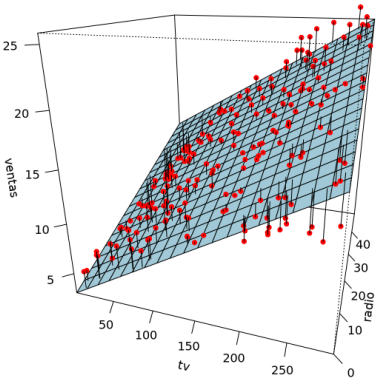
- **Propósito**: Modelar una relación **lineal** entre la variable dependiente y y **varias variables independientes** (o características) $x_1, x_2, \ldots, x_n$.
  La relación es una extensión de la lineal en un espacio **multidimensional.**
- **Ecuación General:**

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- **Características**:
  - Cada xi representa una **característica distinta** o variable independiente.
  - La relación entre y y cada xi es **lineal**. Es decir, el efecto de cada xi en y se representa con un término θixi, sin potencias ni combinaciones de xi.
  - **No incluye términos no lineales** de cada x, como x^2 o x^3.
- **Clase LinearRegression:**
- **Ejemplo**: Predecir el precio de una casa en función de variables como el tamaño (metros cuadrados), el número de habitaciones y la edad de la casa. Cada variable independiente tiene un efecto lineal en el precio.
- **Visualización**:

Regresión Linear Multiple 2D (grado 2) / Regresión Linear Multiple 3D (grado 3)





- **Ejemplo código Python:**

```
import       as
from                    import
from                       import
from                import

# Datos de ejemplo
# Variables independientes (e.g., tamaño y número de habitaciones)

     50, 1    # Tamaño: 50 m², Habitaciones: 1
     60, 2
     75, 2
     80, 3
     100, 4
     120, 4


# Variable dependiente (e.g., precio del inmueble)
             150000  180000  250000  300000  400000  450000

# Dividir los datos en conjunto de entrenamiento y prueba
                                                                    0.2
             42

# Crear el modelo de regresión lineal


# Entrenar el modelo


# Realizar predicciones


# Evaluar el modelo



print  Coeficientes                # Pendientes para cada variable
independiente
print  Intercepto
print  Error Cuadrático Medio  MSE
print  Coeficiente de Determinación (R²):",

# Predicción para un nuevo dato
                        85  3     # Tamaño: 85 m², Habitaciones: 3

print f Predicción para un inmueble de             0  0   m² y
            0  1   habitaciones              0

#Resultado

Coeficientes: [ 2681.15942029 40942.02898551]
Intercepto: -34420.28985507239
Error Cuadrático Medio (MSE): 445759819.3656788
```

```
Coeficiente de Determinación (R²): -0.9811547527363502
Predicción para un inmueble de 85 m² y 3 habitaciones: 316304.3478260869
```

2.2. Regresión Polinomial

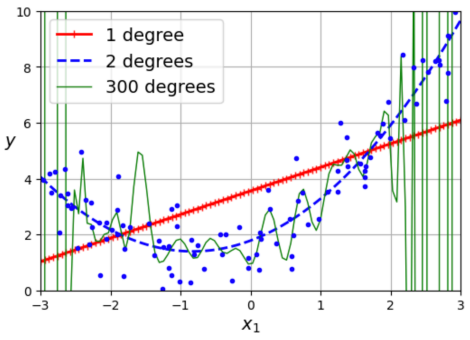
- **Propósito:** Modelar una relación no lineal entre la variable dependiente y y una o más variables independientes mediante el uso de **potencias** de las variables independientes. Extiende la regresión lineal añadiendo términos polinómicos.
- **Ecuación General:**
  - **(Polinomio de grado 2)**:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$

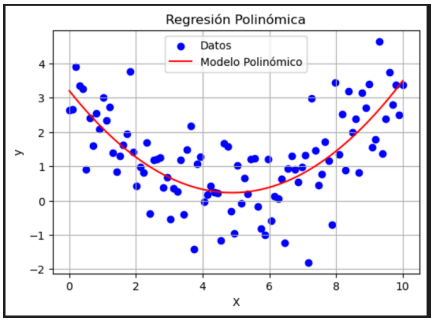
  - **(Polinomio de grado nnn)**:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^n$$

- **Características**:
  - La regresión polinómica **incluye potencias** de la misma variable x (por ejemplo, x, x, x^2, x^3 , etc.) para capturar relaciones **no lineales** entre x e y.
  - Si la regresión polinómica se usa con más de una variable independiente, podría incluir términos como $x_1^2, x_2^2, x_1 \cdot x_2,$, etc., para capturar interacciones entre las variables.
  - Permite que el modelo ajuste **curvas** en lugar de solo líneas rectas.
  - Cuando hay múltiples características, la regresión polinomial es capaz de encontrar relaciones entre características
  - La importancia de los grados. Este modelo de regresión polinomial está sobreajustando mucho los datos de entrenamiento, mientras que el modelo lineal los está subajustando



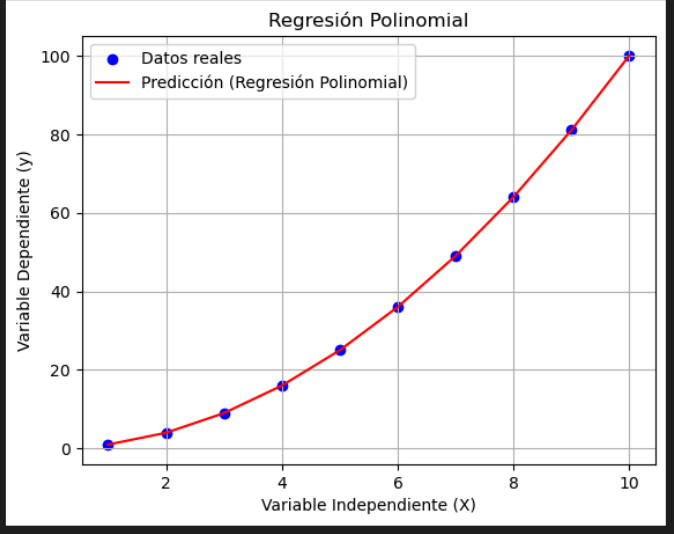
- **Ejemplo**: Predecir la altura de una planta en función del tiempo (días). Si el crecimiento de la planta sigue una relación cuadrática o cúbica, usar un polinomio de grado 2 o 3 puede ayudar a capturar mejor esa relación no lineal.
- **Clase PolynomialFeatures:** de Scikit-Learn para transformar nuestros datos de entrenamiento, añadiendo el cuadrado (polinomio de segundo grado) de cada característica en el conjunto de entrenamiento como una característica nueva (en este caso, solo hay una característica)
- **Visualización**:



- **Ejemplo código Python:**

```
import       as
import                  as
from                      import
from                    import
from                import

# Datos de ejemplo (relación no lineal)
              1    2    3    4    5    6    7    8    9    10
             1  4  9  16  25  36  49  64  81  100


# Crear características polinómicas de grado 2



# Crear y entrenar el modelo de regresión lineal con características
polinómicas



# Predicciones


# Evaluar el modelo



print  Coeficientes
print  Intercepto
print  Error Cuadrático Medio  MSE
print  Coeficiente de Determinación

# Visualizar los resultados
                                    'Datos reales'
                                     'Predicción (Regresión Polinomial)'
          'Regresión Polinomial'
           'Variable Independiente
           'Variable Dependiente




#Resultado
```

```
Coeficientes: [ 0.00000000e+00 -1.16123997e-15  1.00000000e+00]
Intercepto: 7.105427357601002e-15
Error Cuadrático Medio (MSE): 7.696324206562496e-30
Coeficiente de Determinación (R²): 1.0
```



- **Validación cruzada:** La validación cruzada es una técnica fundamental en modelos de regresión para evaluar su capacidad de generalización y evitar problemas como el sobreajuste o subajuste. En este contexto, el objetivo es medir qué tan bien el modelo predice valores continuos en datos no vistos
  - **Tipos Comunes de Validación Cruzada**:
    - **K-Fold Cross-Validation**:
      - Divide los datos en K subconjuntos (folds).
      - Entrena el modelo en K−1 folds y lo valida en el fold restante.
      - Repite este proceso KKK veces y promedia los resultados.
    - **Leave-One-Out Cross-Validation (LOOCV)**:
      - Usa un solo ejemplo como validación y entrena en el resto de los datos.
  - **Propósito**:
    - Evaluar la capacidad del modelo para generalizar a nuevos datos.
    - Detectar sobreajuste o subajuste sin depender de una única partición fija.
  - **Herramienta en scikit-learn**:
    - La función cross_val_score() facilita la implementación de validación cruzada:
  - Utilizando la validación cruzada podemos obtener una estimación del rendimiento de la generalización de un modelo:
    - Si un modelo tiene un buen rendimiento en los datos de entrenamiento, pero generaliza mal según las métricas de validación cruzada, entonces está sobreajustando.
    - Si tiene un mal rendimiento en ambos casos, entonces está subajustando.
  - **Codigo ejemplo Python:**

```
# Ejemplo K-Fold Cross-Validation

from                          import
from                       import
import        as

# Datos de ejemplo
                1     2     3     4     5     6     7     8     9     10      # Variable
independiente
```

```
              1.5, 2.5, 3.0, 4.0, 5.5, 6.5, 7.0, 8.0, 9.5, 10.5    # Variable
dependiente

# Modelo


# Validación cruzada con 5 folds
                                          5
print  Errores Cuadráticos Medios por Fold
print      Promedio



# Ejemplo Leave-One-Out Cross-Validation (LOOCV)

from                         import




print
```

- **Curvas de aprendizaje:** Las curvas de aprendizaje son gráficos que muestran cómo varía el error de entrenamiento y el error de validación del modelo a medida que se entrena. Estos gráficos permiten identificar si un modelo está **sobreajustado** o **subajustado**.

  - **¿Cómo se construyen?**:
    - Evalúa el modelo en intervalos regulares durante el entrenamiento, tanto en el conjunto de entrenamiento como en el de validación.
    - Traza el gráfico de los errores obtenidos (e.g., error cuadrático medio o precisión).
  - **Propósito**:
    - Identificar **sobreajuste**: Ocurre cuando el error de entrenamiento es mucho menor que el error de validación.
    - Identificar **subajuste**: Ocurre cuando ambos errores son altos y el modelo no captura adecuadamente la complejidad de los datos.
  - **Herramienta en scikit-learn**:
    - La función learning_curve() entrena y evalúa el modelo en subconjuntos crecientes del conjunto de entrenamiento y genera datos para trazar las curvas.
    - Ejemplo básico python:

```
from                          import
import                    as

# Datos y modelo
            # Tus datos
             # Tu modelo (e.g., LinearRegression())

# Obtener resultados

   5

# Graficar las curvas
                                                          Error de
Entrenamiento
                                                        Error de
Validación

           Curvas de Aprendizaje
            Tamaño del conjunto    entrenamiento
            Precisión
```

2.3. Regresión Logística

Regresión Logística (aunque no es exactamente regresión en el sentido clásico): Se usa para problemas de clasificación binaria, donde la variable dependiente es categórica. Modela la probabilidad de pertenencia a una categoría.

2.4. Tabla comparativa de Regresiones

Aspecto	Regresión Lineal Simple	Regresión Lineal Múltiple	Regresión Polinómica
Tipo de Relación	Lineal	Lineal	No lineal
Variables Independientes	Una sola variable x	Varias variables $x_1, x_2, \ldots, x_n$	Una o más variables elevadas a potencias
Ecuación	$y = \theta_0 + \theta_1 x$	$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$	$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_n x^n$
Complejidad	Baja	Moderada	Alta
Ejemplo de Uso	Predecir precio en función de una sola característica	Predecir precio en función de varias características	Modelar crecimiento o relaciones no lineales
Limitaciones	Solo una variable, no captura relaciones complejas	No captura relaciones no lineales	Mayor riesgo de sobreajuste con grados altos

Resumen de las Diferencias

- **Regresión Lineal Simple**: Usa una sola variable independiente con una relación lineal. Es el modelo más básico y fácil de interpretar.
- **Regresión Lineal Múltiple**: Extiende la regresión lineal simple para incluir varias variables independientes, pero mantiene una relación lineal con cada variable.
- **Regresión Polinómica**: Introduce potencias de las variables para capturar relaciones no lineales, permitiendo modelar patrones más complejos, aunque con mayor riesgo de sobreajuste si el grado es alto.

2.5. Flujo de Trabajo con Técnicas de Optimización y Ajuste en Regresión Lineal

1. **Inicia con una Regresión Lineal Básica**:
   - En general, comienzas con una regresión lineal simple o múltiple para ver si el modelo ajusta bien los datos y proporciona buenos resultados.
   - Realiza una primera evaluación para ver los errores de entrenamiento y validación y analizar si el modelo generaliza bien o si hay indicios de sobreajuste o subajuste.
2. **Aplicación de Métodos de Optimización**:
   - Si notas que el modelo no está ajustando bien, puedes **utilizar métodos de optimización** como el **descenso de gradiente por lotes** o el **descenso de gradiente estocástico** para ajustar los coeficientes.
   - Estos métodos iterativos pueden mejorar el ajuste, especialmente cuando se trabaja con conjuntos de datos grandes y no se puede usar un método exacto (como una solución cerrada) debido a problemas de rendimiento.
   - **SGD** es particularmente útil si tienes muchos datos o necesitas un modelo que se ajuste rápidamente en aplicaciones en tiempo real.
3. **Aplicación de la Descomposición de Valores Singulares (SVD)**:

- Si el conjunto de datos tiene características **altamente correlacionadas** (multicolinealidad) o si estás trabajando con una matriz muy grande y compleja, la SVD puede ayudar a estabilizar la solución.
- La SVD descompone los datos en componentes principales, permitiendo obtener una solución más estable y reducir la dimensionalidad si es necesario.

4. **Ajuste de Regularización si es Necesario**:
   - Si después de entrenar el modelo aún ves problemas de sobreajuste (el modelo funciona bien en entrenamiento, pero mal en validación), puedes añadir **regularización** (como Ridge o Lasso) para penalizar los coeficientes grandes y reducir la complejidad del modelo.
   - La regularización también ayuda a mejorar el rendimiento del modelo en datos nuevos al evitar que el modelo se ajuste demasiado al conjunto de datos de entrenamiento.

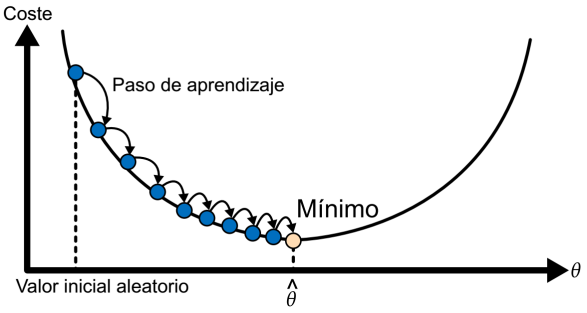
3. Métodos de optimización

Los **métodos de optimización** son algoritmos matemáticos diseñados para encontrar los valores óptimos de los parámetros de un modelo, de modo que se minimice (o maximice) una **función objetivo** o **función de pérdida**. En el contexto de la **regresión lineal** y de muchos otros modelos de machine learning, los métodos de optimización se utilizan para encontrar los coeficientes o pesos que minimizan el error de predicción de forma iterativa.

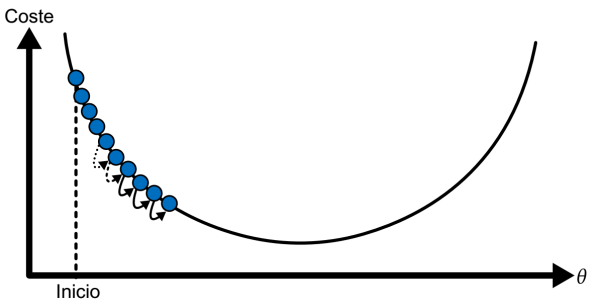
Descenso de Gradiente:

El descenso de gradiente es un método iterativo para minimizar una función de pérdida ajustando los parámetros del modelo (como los coeficientes en regresión lineal). Funciona calculando la dirección del gradiente (derivada parcial) y actualizando los parámetros en la dirección opuesta al gradiente para reducir el error.

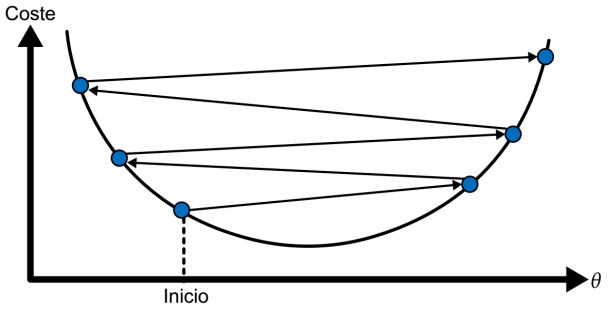
La idea general del descenso de gradiente es ajustar parámetros de forma iterativa para minimizar la función de pérdida. En la práctica, se empieza por rellenar θ con valores aleatorios (esto se denomina inicialización aleatoria). Después, se mejora de manera gradual, pasito a pasito, intentando con cada paso reducir la función de pérdida (por ejemplo, el ECM), hasta que el algoritmo converge en un mínimo.



Un parámetro importante en el descenso de gradiente es el tamaño de los pasos (el hiperparámetro tasa de aprendizaje). Si la tasa de aprendizaje es demasiado pequeña, entonces el algoritmo tendrá que pasar por muchas iteraciones para converger, lo cual llevará mucho tiempo.



Por otra parte, si la tasa de aprendizaje es demasiado alta, puede que saltes por encima del valle y acabes al otro lado, posiblemente más alto de lo que estabas antes. Esto podría hacer que el algoritmo diverja, con valores cada vez más grandes, y no consiga encontrar una buena solución.



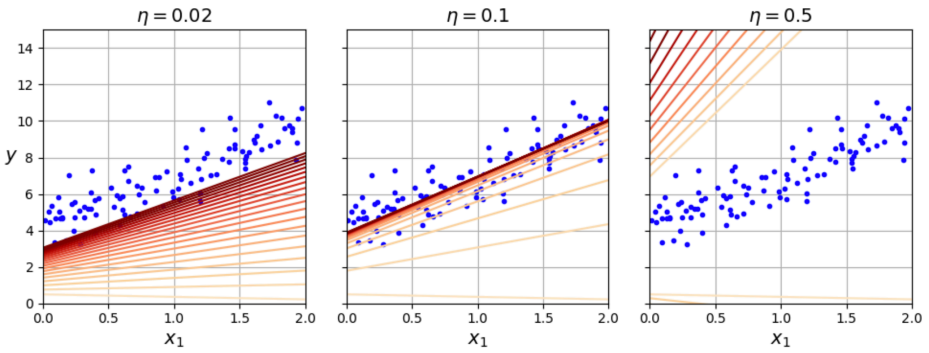
Por suerte, está garantizado que aplicando el descenso de gradiente sobre la función de pérdida del ECM se queda arbitrariamente cerca del mínimo global (si esperamos lo suficiente y si la tasa de aprendizaje no es demasiado alta), puesto que la función de perdida tiene forma de bol.

Cuantos más parámetros tenga un modelo, más dimensiones tendrá este espacio y más difícil será la búsqueda.

Variantes de Descenso de Gradiente:

1. **Descenso de gradiente por lotes - Batch Gradient Descent**:
    - Utiliza todo el conjunto de datos para calcular el gradiente.
    - Es computacionalmente costoso para grandes conjuntos de datos.
    - Necesitamos calcular cuánto cambiará la función de pérdida si cambiamos θ_j solo un poquito (derivada parcial).
    - El vector de gradiente contiene todas las derivadas parciales de la función de pérdida (una para cada parámetro del modelo). Esto implica cálculos sobre el conjunto de entrenamiento completo X y como resultado, es lentísimo en conjuntos de entrenamiento muy grandes.
    - El descenso de gradiente escala bien con el número de características.
    - **Clase SGDRegressor**, que tiende por defecto a optimizar la función de pérdida del ECM.
    - Ejemplo:

La siguiente figura muestra los 20 primeros pasos del descenso de gradiente utilizando tres tasas de aprendizaje diferentes. La línea de la parte inferior de cada gráfico representa el punto de partida aleatorio y, después, cada repetición se representa con una línea cada vez más oscura.



A la izquierda, la tasa de aprendizaje es demasiado baja: el algoritmo acabará llegando a la solución, pero tardará mucho. En el centro, la tasa de aprendizaje tiene un aspecto bastante bueno: en solo unas repeticiones, ya ha convergido en la solución. A la derecha, la tasa de aprendizaje es demasiado alta: el algoritmo diverge, salta por todas partes y, en realidad, se aleja más y más de la solución con cada paso.

- Ejemplo en Python:

```
import       as

# Datos de ejemplo (X: cantidad de unidades vendidas, y: precio del producto)
              1    2    3    4    5          float
             2.5  4.5  6.5  8.5  10.5         float   # Relación lineal y = 2
* X + 0.5

# Añadir el término de sesgo (columna de 1s para intercepto)
                                        # X_b = [[1, x1], [1, x2], ...]

# Inicialización de parámetros
                      2  1   # [intercepto, pendiente]
                0.01
              1000
           # Número de muestras

# Gradiente descendente por lotes
                range
                2                                                  -1  1
                                                 # Actualización de parámetros

# Mostrar parámetros finales
print  Intercepto          0  0
print  Pendiente          1  0

# Predicciones

print  Predicciones
```

2. **Descenso del gradiente Estostatico - Stochastic Gradient Descent (SGD)**:
   - Actualiza los parámetros utilizando una sola muestra de datos a la vez.
   - Más rápido pero menos estable.
   - Elige una instancia aleatoria en el conjunto de entrenamiento a cada paso y calcula los gradientes basándose solo en esa única instancia
   - Es más rápido y agíl con conjuntos de entrenamiento enormes

- Lo malo, es que debido a su naturaleza aleatoria, es mucho menos regular que el descenso por lotes.
- **Clase SGDRegressor**, que tiende por defecto a optimizar la función de pérdida del ECM.
- Ejemplo en Python

```
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import train_test_split
import numpy as np

# Datos de ejemplo
X = np.random.rand(100, 1) * 10
y = 3 * X.flatten() + np.random.randn(100) * 2  # Relación lineal con ruido

# Dividir datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Modelo SGD
sgd = SGDRegressor(max_iter=1000, learning_rate='constant', eta0=0.01,
random_state=42)
sgd.fit(X_train, y_train)
y_pred = sgd.predict(X_test)

# Mostrar coeficientes
print("Coeficientes:", sgd.coef_)
print("Intercepto:", sgd.intercept_)

# Resultados
print("Coeficiente (Pendiente):", sgd.coef_[0])
print("Intercepto:", sgd.intercept_[0])
print("Error Cuadrático Medio (MSE):", mean_squared_error(y, y_pred))

# Predicción para un nuevo dato
nuevo_valor = np.array([[6]], dtype=float)
nuevo_valor_scaled = scaler.transform(nuevo_valor)
prediccion = sgd.predict(nuevo_valor_scaled)
print(f"Predicción para {nuevo_valor[0][0]} unidades vendidas:",
prediccion[0])
```

Aplicación en Regresión:

El descenso de gradiente se utiliza para optimizar funciones de pérdida como el error cuadrático medio (MSE). Es común en modelos como la regresión lineal y regularizada (Ridge, Lasso).

3. Descomposición de Valores singulares (SVD - Singular Value Decomposition)

Es una técnica de algebra lineal que descompone una matriz en tres matrices componentes y es la técnica usada por Scikit-Learn para calcular la regresión lineal con su algoritmo LinearRegression.

Sirve para:

- **Resolución de Problemas de Colinealidad**: La SVD proporciona soluciones estables en regresión lineal al manejar datos donde las variables independientes están altamente correlacionadas.
- **Reducción Dimensional y Optimización**: Es clave para simplificar datos complejos manteniendo la mayor cantidad de información posible, utilizada en análisis y compresión de datos.
- **Soluciones Directas y Eficientes**: Permite encontrar soluciones exactas de regresión lineal en una sola operación, sin necesidad de iteraciones, ahorrando tiempo y recursos computacionales.
- **Ejemplo de Python básico:**

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

```python
from sklearn.metrics import mean_squared_error, r2_score

# Datos de ejemplo (X: cantidad de unidades vendidas, y: precio del
producto)
X = np.array([[1], [2], [3], [4], [5]])  # Variable independiente
y = np.array([2.5, 4.5, 6.5, 8.5, 10.5])  # Variable dependiente

# Crear el modelo de regresión lineal
model = LinearRegression()

# Entrenar el modelo
model.fit(X, y)

# Predicción de valores
y_pred = model.predict(X)

# Resultados
print("Coeficiente:", model.coef_[0])  # Pendiente de la recta
print("Intercepto:", model.intercept_)  # Punto donde la recta cruza el
eje Y

# Evaluación del modelo
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

print("Error Cuadrático Medio (MSE):", mse)
print("Coeficiente de Determinación (R²):", r2)

# Reconstrucción con SVD (Internamente usado por LinearRegression)
U, S, Vt = np.linalg.svd(X, full_matrices=False)
Sigma = np.diag(S)
X_reconstructed = U @ Sigma @ Vt

print("\nReconstrucción de X (con SVD):")
print(X_reconstructed)
```

¿En Qué Parte del Proceso se Usan los Métodos de Optimización?

Los métodos de optimización se aplican durante la fase de **entrenamiento del modelo**. Este es el proceso general:

1. **Definición de la Función de Pérdida**:
    - Primero, se define una función que mida el error del modelo. En la regresión lineal, comúnmente se utiliza el **error cuadrático medio (MSE)**, que calcula la diferencia entre los valores predichos y los valores reales al cuadrado y luego promedia estos errores.
2. **Inicialización de Parámetros**:
    - Los parámetros del modelo (los coeficientes en una regresión lineal, por ejemplo) se inicializan. Pueden empezar con valores aleatorios o con ceros.
3. **Optimización**:
    - Aquí es donde entran los métodos de optimización. Su objetivo es **ajustar los parámetros del modelo** para minimizar la función de pérdida.
    - El optimizador ajusta iterativamente los parámetros hasta que el modelo se ajuste de la mejor manera posible a los datos de entrenamiento.
    - Esto se logra calculando el **gradiente** de la función de pérdida con respecto a los parámetros y actualizando los parámetros en la dirección opuesta al gradiente para reducir el error (en métodos como el descenso de gradiente).

Resumen del Proceso

1. **Inicio del Entrenamiento**: Se define una función de pérdida que mide el error del modelo.

2. **Optimización**: Se aplica un método de optimización que ajusta los parámetros del modelo iterativamente para reducir el error de la función de pérdida.
3. **Convergencia**: El método de optimización finaliza cuando se ha alcanzado un mínimo de error (o un número de iteraciones determinado), y el modelo está ajustado para hacer predicciones.

Comparación entre tres métodos de optimización para resolver problemas de regresión lineal:

Algoritmo	m grande (número de filas)	n grande (número de columnas)	Hiperparámetros	Escalado requerido	Scikit-Learn
Descomposición de valores singulares	Rápido	Lento	0	No	LinearRegression
Descenso de gradiente por lotes	Lento	Rápido	2	Si	SGDRegressor
Descenso de gradiente estocástico	Rápido	Rápido	≥ 2	Si	SGDRegressor

Casi no hay diferencia después del entrenamiento: todos estos algoritmos acaban con modelos muy similares.

4. Regularización

La **regularización** en la **regresión** es una técnica utilizada para **evitar el sobreajuste** (overfitting) al penalizar los valores grandes de los coeficientes del modelo. Esto se logra agregando un término de penalización a la **función de pérdida** que el modelo intenta minimizar, lo cual limita la complejidad del modelo y le permite generalizar mejor en datos nuevos.

En términos simples, la regularización controla el tamaño de los coeficientes:

- Un modelo sin regularización puede ajustarse demasiado a los datos de entrenamiento, capturando ruido y patrones irrelevantes.
- Con la regularización, se evita que los coeficientes crezcan mucho, lo que simplifica el modelo y reduce la probabilidad de que se ajuste a variaciones aleatorias en los datos de entrenamiento.

Tipos de Regularización en Regresión

- **Ridge (L2)**: Agrega una penalización de tipo L2, que es la suma de los cuadrados de los coeficientes. Esto reduce los valores de los coeficientes, pero sin eliminarlos completamente.
- **Lasso (L1)**: Agrega una penalización de tipo L1, que es la suma de los valores absolutos de los coeficientes. Esta penalización puede reducir algunos coeficientes a cero, eliminando características irrelevantes y ayudando en la selección de características.
- **Elastic Net**: Combina ambas penalizaciones L1 y L2, logrando un balance entre la reducción de los coeficientes y la eliminación de características.

Objetivo de la Regularización

La regularización busca **mejorar la capacidad de generalización del modelo** al reducir el riesgo de sobreajuste, lo que permite que el modelo sea más robusto y preciso en datos que no ha visto antes.

4.1. Regularización Ridge(L2)

Resumen

La clave para utilizar la regularización **Ridge** correctamente es encontrar el valor óptimo de α que minimice tanto el error de entrenamiento como el de validación. Esto se puede lograr utilizando **validación cruzada** para probar diferentes valores de α y evitar tanto el **sobreajuste** como el **subajuste**.

Características:

- El término de regularización solo debería añadirse a la función de pérdida durante el entrenamiento.
- Una vez que el modelo está entrenado, conviene utilizar el ECM (o la RECM) no regularizado para evaluar el rendimiento del modelo
- El hiperparámetro α controla cuánto queremos regularizar el modelo
- Es importante escalar los datos(por ejemplo, utilizando StandardScaler) antes de realizar la regresión de Ridge ya que es sensible a la escala de las características de entrada

Código ejemplo Python:

```python
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

# Datos de ejemplo
X = [[1], [2], [3], [4], [5]]  # Variable independiente
y = [2.5, 4.5, 6.5, 8.5, 10.5]  # Variable dependiente

# Modelo Ridge
ridge = Ridge(alpha=1.0)  # El parámetro alpha controla la regularización
ridge.fit(X, y)

# Predicción
y_pred = ridge.predict(X)

# Resultados
print("Coeficiente (Pendiente):", ridge.coef_[0])
print("Intercepto:", ridge.intercept_)
print("MSE:", mean_squared_error(y, y_pred))
```

```
# Salida consola
Coeficiente (Pendiente): 1.8181818181818181
Intercepto: 1.0454545454545459
MSE: 0.06611570247933886
```

Tabla comparativa de posibles escenarios

Aquí tienes una **tabla comparativa** para la regularización **Ridge (L2)**, donde se muestran los posibles escenarios en función del valor de α, las observaciones sobre el error, y la acción recomendada sobre α para evitar **sobreajuste** o **subajuste**.

Escenario	Valor de α	Observación del Error	Estado del Modelo	Acción Recomendada	Riesgo Potencial
Sin Regularización	α=0	Error de entrenamiento bajo, error de validación alto	Sobreajuste	Aumentar α para reducir la complejidad	Sobreajuste
Regularización Moderada	Valor bajo de α	Error de entrenamiento y validación equilibrados y bajos	Buen ajuste	Mantener o ajustar ligeramente según los datos	Ninguno

Escenario	Valor de α	Observación del Error	Estado del Modelo	Acción Recomendada	Riesgo Potencial
Regularización Moderada-Alta	Valor medio de α	Error de entrenamiento y validación similares pero algo altos	Ligeramente subajustado	Reducir α para mejorar el ajuste	Subajuste
Regularización Alta	Valor alto de α	Error de entrenamiento y validación similares y altos	Subajuste	Reducir α para permitir mayor complejidad	Subajuste
Regularización Muy Alta	$\alpha \to \infty$	Error de entrenamiento y validación muy altos, modelo casi plano	Fuerte Subajuste	Reducir α significativamente	Subajuste severo

Explicación de la Tabla

- **Sin Regularización (α=0)**:
  - Sin regularización, el modelo no tiene restricciones en los coeficientes, lo que puede llevar a un sobreajuste, especialmente si el modelo es complejo.
  - **Acción**: Aumentar α para controlar los coeficientes y reducir el sobreajuste.
- **Regularización Moderada (Valor Bajo de α)**:
  - Con una regularización ligera, el modelo puede estar bien ajustado si el error de entrenamiento y el de validación son bajos y equilibrados.
  - **Acción**: Mantener el valor de α o ajustarlo ligeramente si se observan cambios en el error.
- **Regularización Moderada-Alta (Valor Medio de α)**:
  - Con un valor medio de α, el modelo puede tener un ligero subajuste si el error es similar entre entrenamiento y validación, pero un poco alto.
  - **Acción**: Reducir α ligeramente para mejorar el ajuste.
- **Regularización Alta (Valor Alto de α)**:
  - Con un valor alto de α, el modelo tiende a subajustarse, ya que los coeficientes están fuertemente penalizados, lo que limita su capacidad de ajustarse a los datos.
  - **Acción**: Reducir α para permitir que el modelo capture más complejidad en los datos.
- **Regularización Muy Alta ($\alpha \to \infty$)**
  - Con un α extremadamente alto, el modelo se convierte casi en una línea plana, resultando en un subajuste severo.
  - **Acción**: Reducir α significativamente, ya que el modelo es demasiado simple y no captura patrones relevantes en los datos.

4.2. Regularización Lasso(L1)

Resumen

Lasso (L1) agrega una penalización basada en la suma de los valores absolutos de los coeficientes. Esto puede llevar algunos coeficientes a ser exactamente cero, eliminando características irrelevantes y ayudando con la selección de características.

Características:

- Añade un término de regularización a la función de pérdida (en este caso al termino de regularización se le conoce como L1)
- Tiende a eliminar los pesos de las características menos importantes (es decir, los pone a cero).

Código ejemplo Python:

```python
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

# Modelo Lasso
lasso = Lasso(alpha=0.1)  # El parámetro alpha controla la regularización
lasso.fit(X, y)

# Predicción
y_pred = lasso.predict(X)

# Resultados
print("Coeficiente (Pendiente):", lasso.coef_[0])
print("Intercepto:", lasso.intercept_)
print("MSE:", mean_squared_error(y, y_pred))
```

Salida consola

Coeficiente (Pendiente): 1.95

Intercepto: 0.6500000000000004

MSE: 0.005000000000000036

Tabla comparativa de posibles escenarios

Escenario	Valor de α	Observación del Error	Estado del Modelo	Acción Recomendada	Riesgo Potencial
Sin Regularización	α=0	Error de entrenamiento bajo, error de validación alto	Sobreajuste	Aumentar α para reducir la complejidad y eliminar características irrelevantes	Sobreajuste
Regularización Moderada	Valor bajo de α	Error de entrenamiento y validación equilibrados y bajos	Buen ajuste	Mantener o ajustar ligeramente según los datos	Ninguno
Regularización Moderada-Alta	Valor medio de α	Error de entrenamiento y validación similares, pero algo altos	Ligeramente subajustado	Reducir α para mejorar el ajuste	Subajuste
Regularización Alta	Valor alto de α	Error de entrenamiento y validación similares y altos, varios coeficientes en cero	Subajuste	Reducir α para permitir mayor complejidad y evitar eliminación excesiva de características	Subajuste
Regularización Muy Alta	α→∞	Error de entrenamiento y validación muy altos, la mayoría de los coeficientes en cero	Fuerte Subajuste	Reducir α significativamente	Subajuste severo

Explicación de la Tabla Lasso

- **Sin Regularización (**α=0**)**:
  - Sin regularización (α=0), el modelo no elimina ninguna característica, lo que puede llevar al sobreajuste si hay variables irrelevantes.
  - Acción: Aumentar α para reducir la complejidad y eliminar características irrelevantes
- **Regularización Moderada (**Valor bajo de α**)**:
  - Un valor bajo de α permite eliminar algunas características irrelevantes mientras mantiene un buen ajuste.
  - Acción: Mantener o ajustar ligeramente según los datos
- **Regularización Moderada-Alta (**Valor medio de α**)**:
  - Si el error de entrenamiento y validación son similares, pero algo altos, reducir α permite un ajuste más preciso.
  - Acción: Reducir α para permitir mayor complejidad y evitar eliminación excesiva de características
- **Regularización Alta (**Valor alto de α**)**:
  - Un valor alto de α puede eliminar demasiadas características, causando subajuste.
  - Acción: Reducir α para permitir mayor complejidad y evitar eliminación excesiva de características
- **Regularización Muy Alta (**α→∞**)**:
  - Un α muy alto elimina la mayoría de los coeficientes, causando subajuste severo.
  - Acción: Reducir α significativamente

4.3. Regularización Elastic Net

Resumen

Elastic Net combina las penalizaciones de Lasso (L1) y Ridge (L2), controladas por dos hiperparámetros: α, que determina la magnitud de la regularización, y r, que define la proporción entre L1 y L2 (por ejemplo, si r=0.5, ambos tipos de regularización se aplican en la misma medida).

Gestión del hiperparámetros R

En **Elastic Net**, el hiperparámetro **r** (a veces llamado **ratio** o **mezcla**) controla la combinación de regularización **L1 (Lasso)** y **L2 (Ridge)**. Su valor puede variar entre **0 y 1**, donde:

- **r=0**: El modelo se comporta como una **regularización Ridge pura** (L2). Esto significa que la penalización solo considera la suma de los cuadrados de los coeficientes, sin eliminar características.
- **r=1**: El modelo se comporta como una **regularización Lasso pura** (L1). Esto significa que la penalización se basa en la suma de los valores absolutos de los coeficientes, lo que puede llevar algunos coeficientes a cero, eliminando características irrelevantes.
- **Valores intermedios (0 < r < 1)**: El modelo aplica una **combinación de L1 y L2**, con una proporción específica entre ambas regularizaciones. Por ejemplo:
    - Si r=0.5, se aplica una mezcla equilibrada de regularización L1 y L2.
    - Si r=0.25, el modelo se inclina más hacia Ridge (L2) que hacia Lasso (L1).
    - Si r=0.75, el modelo se inclina más hacia Lasso (L1) que hacia Ridge (L2).

¿Cuándo usar cada rango de r?

- **r cercano a 0** (más parecido a Ridge): Útil cuando tienes muchas variables correlacionadas y quieres mantener todas las características en el modelo sin eliminarlas.
- **r cercano a 1** (más parecido a Lasso): Útil cuando quieres realizar una **selección de características**, es decir, eliminar las variables menos importantes estableciendo algunos coeficientes en cero.
- **Valores intermedios**: Útiles cuando tienes datos complejos y necesitas una combinación de ambos métodos: la capacidad de eliminar características irrelevantes (L1) y la estabilidad frente a características correlacionadas (L2).

Código ejemplo Python

```
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error

# Modelo ElasticNet
elasticnet = ElasticNet(alpha=0.1, l1_ratio=0.5)  # l1_ratio controla el balance
entre L1 y L2
elasticnet.fit(X, y)

# Predicción
y_pred = elasticnet.predict(X)

# Resultados
print("Coeficiente (Pendiente):", lasso.coef_[0])
print("Intercepto:", lasso.intercept_)
print("MSE:", mean_squared_error(y, y_pred))


# Salida consola
Coeficiente (Pendiente): 1.95
Intercepto: 0.6500000000000004
MSE: 0.010707911957168501
```

Tabla comparativa de posibles escenarios

Escenario	Valor de α y r	Observación del Error	Estado del Modelo	Acción Recomendada	Riesgo Potencial
Sin Regularización	α=0	Error de entrenamiento bajo, error de validación alto	Sobreajuste	Aumentar α para reducir complejidad y eliminar características irrelevantes	Sobreajuste
Regularización Moderada	Valor bajo de α, r moderado	Error de entrenamiento y validación equilibrados y bajos	Buen ajuste	Mantener o ajustar ligeramente según los datos	Ninguno
Regularización Moderada-Alta	Valor medio de α, r moderado	Error de entrenamiento y validación similares pero algo altos	Ligeramente subajustado	Reducir α para mejorar el ajuste	Subajuste
Regularización Alta	Valor alto de α, r alto (predominante L1)	Error de entrenamiento y validación altos, varios coeficientes en cero	Subajuste	Reducir α y/o ajustar r para reducir la eliminación excesiva de características	Subajuste
Regularización Muy Alta	α→∞, r bajo (predominante L2)	Error de entrenamiento y validación muy altos, coeficientes muy pequeños, pocos en cero	Fuerte Subajuste	Reducir α significativamente y ajustar r para mejorar el ajuste	Subajuste severo

Explicación de la Tabla Elastic Net

- **Sin Regularización (**α=0**)**:
  - Con α=0, no hay ninguna penalización, lo que puede llevar al sobreajuste si el modelo es demasiado complejo.
  - Acción: Aumentar α para reducir complejidad y eliminar características irrelevantes
- **Regularización Moderada (**Valor bajo de α, r moderado**)**:
  - Un valor bajo de α y un valor moderado de r mantienen un buen ajuste y seleccionan algunas características irrelevantes.
  - Acción: Mantener o ajustar ligeramente según los datos
- **Regularización Moderada-Alta (**Valor medio de α, r moderado**)**:
  - Si el modelo muestra subajuste ligero, reducir α permite una mejor generalización.
  - Acción: Reducir α para mejorar el ajuste
- **Regularización Alta (**Valor alto de α, r alto (predominante L1)**)**:
  - Un α alto y un r alto (predominancia de L1) eliminan varias características, lo que puede causar subajuste.
  - Acción: Reducir α y/o ajustar r para reducir la eliminación excesiva de características
- **Regularización Muy Alta (**α→∞, r bajo (predominante L2)**)**:
  - Con α muy alto y r bajo (predominancia de L2), los coeficientes son muy pequeños pero no eliminados, causando un subajuste severo.
  - Acción: Reducir α significativamente y ajustar r para mejorar el ajuste

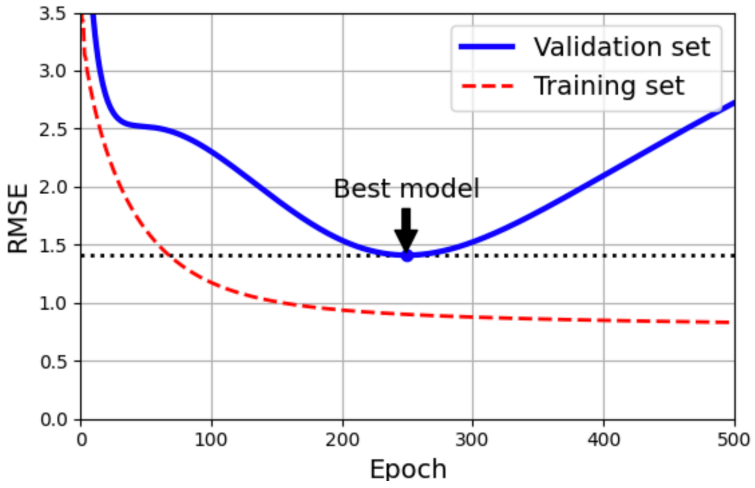
Resumen de Acciones en Ambas Regularizaciones

- **Sobreajuste**: Si ves que el modelo sobreajusta, **aumenta α** (en Ridge, Lasso, o Elastic Net) para penalizar los coeficientes y reducir la complejidad.
- **Subajuste:** Si el modelo subajusta, **reduce** α para que el modelo capture mejor las relaciones. En Elastic Net, puedes también ajustar el valor de r para cambiar la proporción entre L1 y L2 según el problema.

4.4. Detención temprana

Una forma muy diferente de regularizar algoritmos de aprendizaje iterativo, como el descenso de gradiente, es detener el entrenamiento en cuanto el error de validación alcanza un mínimo. Esto se denomina detención temprana.

La siguiente figura muestra un modelo complejo (en este caso, un modelo de regresión polinomial de alto grado) que se está entrenando con descenso de gradiente por lotes en el conjunto de datos cuadrático que hemos utilizado antes.



A medida que avanzan las repeticiones, el algoritmo aprende y su error de predicción (RECM) en el conjunto de entrenamiento baja, junto con su error de predicción en el conjunto de validación. Sin embargo, después de un tiempo, el error de validación deja de reducirse y empieza a subir otra vez. Eso indica que el modelo ha empezado a sobreajustar los datos de entrenamiento.

Puede parar el entrenamiento en cuanto el error de validación alcanza el mínimo o parar solo después de que el error de validación lleve un tiempo por encima del mínimo.

```
from copy import deepcopy
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

np.random.seed(42)
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m, 1)
X_train, y_train = X[: m // 2], y[: m // 2, 0]
X_valid, y_valid = X[m // 2 :], y[m // 2 :, 0]

preprocessing = make_pipeline(PolynomialFeatures(degree=90, include_bias=False), StandardScaler())
X_train_prep = preprocessing.fit_transform(X_train)
X_valid_prep = preprocessing.transform(X_valid)
sgd_reg = SGDRegressor(penalty=None, eta0=0.002, random_state=42)
n_epochs = 500
```

```
best_valid_rmse = float('inf')
train_errors, val_errors = [], []

for epoch in range(n_epochs):
    sgd_reg.partial_fit(X_train_prep, y_train)
    y_valid_predict = sgd_reg.predict(X_valid_prep)
    val_error = mean_squared_error(y_valid, y_valid_predict,
        squared=False)
    if val_error < best_valid_rmse:
        best_valid_rmse = val_error
        best_model = deepcopy(sgd_reg)

    y_train_predict = sgd_reg.predict(X_train_prep)
    train_error = mean_squared_error(y_train, y_train_predict,
        squared=False)
    val_errors.append(val_error)
    train_errors.append(train_error)

# para dibujar la imagen
best_epoch = np.argmin(val_errors)
plt.figure(figsize=(6, 4))
plt.annotate('Best model',
             xy=(best_epoch, best_valid_rmse),
             xytext=(best_epoch, best_valid_rmse + 0.5),
             ha="center",
             arrowprops=dict(facecolor='black', shrink=0.05))
plt.plot([0, n_epochs], [best_valid_rmse, best_valid_rmse], "k:", linewidth=2)
plt.plot(val_errors, "b-", linewidth=3, label="Validation set")
plt.plot(best_epoch, best_valid_rmse, "bo")
plt.plot(train_errors, "r--", linewidth=2, label="Training set")
plt.legend(loc="upper right")
plt.xlabel("Epoch")
plt.ylabel("RMSE")

plt.axis([0, n_epochs, 0, 3.5])

plt.grid()
```

5. Algoritmos de trabajo Regresión

Algoritmo	Descripción	Clase en Scikit-Learn
Regresión Lineal	Modelo básico de regresión que asume una relación lineal entre la variable dependiente y una o más variables independientes.	`LinearRegression`
Regresión Lineal Múltiple	Extiende la regresión lineal simple para incluir múltiples variables independientes.	`LinearRegression`
Regresión Polinómica	Expande las características con potencias de cada variable independiente para modelar relaciones no lineales.	`PolynomialFeatures` + `LinearRegression`
Descomposición en Valores Singulares (SVD)	Técnica de álgebra lineal para resolver problemas de regresión lineal con matrices mal condicionadas o colineales.	`LinearRegression` (con solver SVD)
Descenso de Gradiente Estocástico	Ajusta los coeficientes iterativamente usando un solo dato de entrenamiento a la vez, lo cual es eficiente para grandes conjuntos de datos.	`SGDRegressor`

Algoritmo	Descripción	Clase en Scikit-Learn
Ridge Regression (L2)	Regresión lineal con regularización L2, que penaliza la suma de los cuadrados de los coeficientes para evitar sobreajuste.	`Ridge`
Lasso Regression (L1)	Regresión lineal con regularización L1, que penaliza la suma absoluta de los coeficientes, haciendo que algunos coeficientes se reduzcan a cero (selección de características).	`Lasso`
Elastic Net	Combina L1 y L2 para regularización, balanceando entre selección de características y estabilidad en la predicción.	`ElasticNet`
Regresión con Support Vector Machines (SVR)	Extensión de las máquinas de soporte vectorial (SVM) para problemas de regresión, permite ajustar márgenes para mejorar la precisión.	`SVR`
Árboles de Decisión para Regresión	Utiliza una estructura de árbol para realizar predicciones basadas en la división de los datos en nodos y ramas.	`DecisionTreeRegressor`
Random Forest	Conjunto de árboles de decisión para mejorar la precisión y reducir el sobreajuste mediante la combinación de múltiples predicciones de árboles individuales.	`RandomForestRegressor`
Gradient Boosting Machines (GBM)	Método de boosting que combina múltiples modelos de árboles débiles para mejorar la precisión de predicción en problemas de regresión.	`GradientBoostingRegressor`
K-Nearest Neighbors (KNN) Regression	Algoritmo basado en la distancia, predice el valor en función de los valores de sus vecinos más cercanos.	`KNeighborsRegressor`

6. Evaluación

La evaluación es un paso crucial en el proceso de regresión. Nos permite medir la calidad del modelo, identificar posibles problemas (como sobreajuste o subajuste) y comparar diferentes enfoques. Las métricas utilizadas dependen del tipo de problema, pero generalmente se enfocan en medir el error entre los valores predichos y los valores reales. **En el caso de la regresión** vamos a predecir **valores no numéricos continuos, en vez de categorías.**

6.1. Métricas Comunes para la Evaluación de Modelos de Regresión

Error Cuadrático Medio (MSE):

- **Detalle**:
  - Mide la media de los cuadrados de los errores (diferencias entre los valores reales y predichos).
  - Los errores graves tienen una **penalización mayor** que con MAE
- **Fórmula**:

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

- **Interpretación**:
  - Cuanto más pequeño, mejor. Sin embargo, puede ser sensible a valores atípicos debido al cuadrado de las diferencias.

Raíz del Error Cuadrático Medio (RMSE):

- **Detalle**:

    - La raíz cuadrada del MSE. Se encuentra en las mismas unidades que la variable dependiente.
    - Los errores graves tienen una penalización mayor que con MAE
    - **Tiene un orden de magnitud cercano a los valores de y (el más popular)**
- **Fórmula**:

$$RMSE = \sqrt{MSE}$$

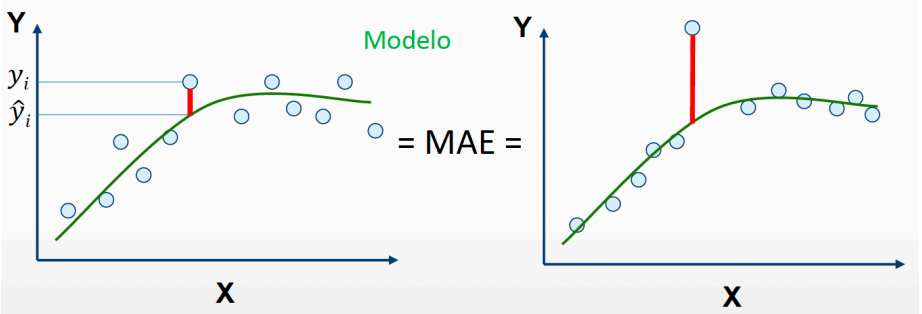
- **Interpretación**:
    - Proporciona una medida intuitiva del error promedio en las mismas unidades que la salida.

Error Absoluto Medio (MAE):

- **Detalle**
    - **No penaliza los errores graves**
    - Mide la media de las diferencias absolutas entre los valores reales y predichos.
    - Sencillo de entender
    - Tiene un orden de magnitud cercano a los valores de y.
- **Fórmula**:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Interpretación**:
    - Es menos sensible a valores atípicos en comparación con el MSE.



6.2. Métodos de Validación

División de Datos en Conjuntos de Entrenamiento y Prueba:

- Se entrena el modelo en una parte de los datos (entrenamiento) y se evalúa en una parte separada (prueba).
- Proporciones comunes: 80% para entrenamiento y 20% para prueba.

Validación Cruzada:

- Divide los datos en múltiples subconjuntos (folds) y evalúa el modelo en cada uno de ellos.
- Tipos:
    - **K-Fold Cross-Validation**: Divide los datos en K subconjuntos y entrena el modelo en K-1 folds mientras valida en el restante.
    - **Leave-One-Out Cross-Validation (LOOCV)**: Usa un solo dato para validar y el resto para entrenar.

Validación por Bootstrapping:

- Genera subconjuntos aleatorios con reemplazo para evaluar la variabilidad del modelo.

6.3. Curvas de Aprendizaje

- Las curvas de aprendizaje muestran cómo varía el error en función del tamaño del conjunto de entrenamiento.
- Permiten identificar:
    - **Sobreajuste**: Error de entrenamiento bajo y error de validación alto.
    - **Subajuste**: Ambos errores altos.

Construcción:
- Dividir el conjunto de entrenamiento en subconjuntos crecientes.
- Entrenar y evaluar el modelo en cada subconjunto.

Visualización:
- Ejes:
    - X: Tamaño del conjunto de entrenamiento.
    - Y: Error (MSE, MAE, etc.).
- Curvas separadas para entrenamiento y validación.

6.4. Comparación entre Modelos

- Para seleccionar el mejor modelo de regresión, se evalúan múltiples métricas y se comparan diferentes enfoques:
    - Regresión Lineal, Ridge, Lasso, Elastic Net, Árboles de Decisión, etc.
- Se prioriza el modelo que tenga:
    - **Buen rendimiento en métricas clave (MSE, R^2, MAE).**
        1. **Seleccionar Métricas Relevantes**:
            - Dependiendo del problema, selecciona las métricas clave que mejor representen el rendimiento del modelo.
                - **Regresión**:
                    1. **MSE**: Útil si los errores grandes son penalizados.
                    2. **MAE**: Adecuado si los errores pequeños y grandes deben ser tratados por igual.
        2. **Calcular las Métricas**:
            - Divide los datos en conjuntos de entrenamiento y prueba.
            - Entrena el modelo en el conjunto de entrenamiento.
            - Predice sobre el conjunto de prueba.
            - Calcula cada métrica usando funciones como mean_squared_error y mean_absolute_error.
        3. **Comparar Modelos**:
            - Entrena múltiples modelos y compara los resultados en las métricas seleccionadas.
            - Prioriza los modelos con:
                - **Menor MSE/MAE**: Indican menor error.
    - **Menor riesgo de sobreajuste o subajuste.**
        1. **Identificar Indicadores de Sobreajuste/Subajuste**:
            - **Sobreajuste**:
                - Bajo error en entrenamiento, alto error en prueba.

- Indicador de que el modelo es demasiado complejo.

- Subajuste**:**
  - Alto error en ambos conjuntos (entrenamiento y prueba).
  - Indicador de que el modelo no captura patrones en los datos.

2. **Evaluar Curvas de Aprendizaje**:

- Entrena el modelo con diferentes tamaños de conjuntos de entrenamiento.
- Calcula y grafica los errores en entrenamiento y validación.
- Identifica si hay una gran discrepancia (sobreajuste) o errores consistentemente altos (subajuste).

3. **Ajustar la Complejidad del Modelo**:

- **Para reducir el sobreajuste:**
  - Regulariza el modelo (e.g., alpha en Ridge/Lasso).
  - Reduce la profundidad de los árboles (max_depth) o ajusta min_samples_split.

- **Para reducir el subajuste:**
  - Aumenta la complejidad del modelo (e.g., degree en polinomios, mayor max_depth).

- **Mejor capacidad de generalización (según validación cruzada).**

1. **Aplicar Validación Cruzada**:
   - Divide los datos en múltiples subconjuntos (folds).
   - Entrena y evalúa el modelo en diferentes particiones, asegurando que cada dato se use tanto para entrenamiento como para prueba.
2. **Obtener Métricas Promedio:**
   - Calcula la métrica seleccionada (e.g., MSE) en cada fold.
   - Promedia los resultados para obtener una métrica representativa del modelo.
3. **Comparar Resultados Entre Modelos:**
   - Prioriza los modelos con menor varianza y mejor desempeño promedio en validación cruzada.
   - Los modelos con alta varianza pueden ser inestables y propensos a sobreajuste.
4. **Usar Técnicas de Validación Avanzada:**
   - **K-Fold Cross Validation**: Usa particiones de datos estándar.
   - **Stratified K-Fold**: Asegura que cada fold tenga una proporción similar de clases (para clasificación).
   - **Leave-One-Out Cross Validation (LOOCV)**: Evalúa el modelo dejando una sola instancia fuera en cada iteración.

6.5. Resumen del Proceso

Criterio	Proceso Clave	Resultado Esperado
Métricas Clave	Calcular MSE, MAE, R^2.	Selección de modelos con menor error y mejor ajuste.
Riesgo de Sobreajuste/Subajuste	Analizar curvas de aprendizaje, ajustar complejidad del modelo.	Modelo balanceado que generalice bien a datos nuevos.
Capacidad de Generalización	Aplicar validación cruzada, obtener métricas promedio y varianza.	Selección del modelo con menor error promedio y baja varianza.

7. Ejemplo Código Python

```
# 1. Generar un conjunto de datos sintético

# Dividir el conjunto de datos en entrenamiento y prueba

# 2. Definir modelos y espacios de búsqueda de hiperparámetros

# 3. Realizar la optimización de hiperparámetros con GridSearchCV

# 4. Evaluar los modelos optimizados

    # Predicciones

    # Calcular métricas
```

```
    # Validación cruzada


    # Guardar resultados




# 5. Curvas de aprendizaje (ejemplo con Ridge Regression optimizado)









# 6. Resumen de resultados
```