UNAL
Mateo Quiceno Zapata
Cristian Ruiz Hernandez

Quiz 1
ML: aprendizaje automático (3010945)

2026-02-16
mquicenoz@unal.edu.co
cruiz@unal.edu.co

# 1. Conjunto de datos

Carga inicial de datos:

```python
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Cargar el dataset
california = fetch_california_housing()
X = california.data
y = california.target
feature_names = california.feature_names

print("Descripción del dataset:")
print(california.DESCR[:1500])
```

## 1.1. Observaciones y características

Ejecute el código anterior e identifique cuántas observaciones (n) y cuántas características tiene el dataset. (3 puntos)

Se considera el siguiente bloque para la solución de los posteriores iterales:

```python
n_observaciones, n_caracteristicas = X.shape
primera_observacion = X[0]

print(f"a) Numero de registros (n): {
    n_observaciones}")

print(f"b) Numero de caracteristicas por
    observacion: {n_caracteristicas}")

print("c) Vector de caracteristicas (x) para
    la primera observacion:")
print(primera_observacion)
print("\n Vector x en forma de columna (
    notacion matematica):")
for i, nombre in enumerate(feature_names):
    print(f" [{primera_observacion[i]:>8.4f}]
        <- {nombre}")
```

### 1.1.1. ¿Cuál es el valor de n (número de registros)?(1 puntos)

Numero de registros (n): 20640

### 1.1.2. ¿Cuántas características tiene cada observación? (1 puntos)

Numero de características por observación: 8

### 1.1.3. Según la notación del capítulo, escriba el vector de características x para la primera observación del dataset en forma de columna. (1 puntos)

$$\mathbf{x}^{(1)} = \begin{bmatrix} 8.3252 \\ 41.0000 \\ 6.9841 \\ 1.0238 \\ 322.0000 \\ 2.5556 \\ 37.8800 \\ -122.2300 \end{bmatrix}$$

## 1.2. Explicación de características

Liste las 8 características del dataset y explique brevemente qué representa cada una. (4 puntos)
Se considera el siguiente bloque para responder:

```python
for i, name in enumerate(california.
    feature_names, 1):
    print(f"{i}. {name}")
desc_text = california.DESCR
start_idx = desc_text.find("Attribute
    Information")
end_idx = desc_text.find("Missing Attribute
    Values")
```

- MedInc El ingreso medio de las familias en el bloque (medido en decenas de miles de dólares).
- HouseAge: La edad mediana de las viviendas en ese bloque.
- AveRooms: El número promedio de habitaciones (incluyendo salas, comedores, etc.) por hogar.
- AveBedrms: El número promedio de dormitorios por hogar.

- Population: El número total de personas que viven en ese grupo de bloques.
- AveOccup: El promedio de personas que viven en una misma casa (densidad por hogar).
- Latitude: La ubicación al norte o sur.
- Longitude: La ubicación al este u oeste.

## 1.3. Variable Y

¿Qué variable estamos tratando de predecir (etiqueta y)? ¿Qué unidades tiene? (2 puntos)

Según la descripción dada por el propio dataset:

```
The target variable is the median house value
    for California districts,
expressed in hundreds of thousands of dollars
    ($100,000).
```

La variable y interesa que describa el valor mediano para una casa en diferentes distritos de California, está se expresa en cientos de miles de dólares.

## 1.4. Regresión o clasificación

Según la taxonomía del capítulo, ¿este problema es de regresión o clasificación? Justifique su respuesta explicando por qué la variable objetivo corresponde a una u otra categoría. (3 puntos)

Este caso es de regresión, la variable que intentamos predecir no pertenece a un rango discreto típico de la clasificación sino a un valor que pertenece a un rango real positivo y por ende no clasificado.

## 1.5. Normalización

Normalización de datos: (8 puntos) El preprocesamiento es crucial para el entrenamiento estable de redes neuronales.

```
# Dividir en conjunto de entrenamiento y
    prueba
X_train, X_test, y_train, y_test =
    train_test_split(
X, y, test_size=0.2, random_state=42)

# Normalizar las características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

### 1.5.1. ¿Por qué es importante normalizar las características antes de entrenar una red neuronal? Relacione su respuesta con el proceso de descenso del gradiente. (3 pts)

Es importante debido a que es requerido que en el proceso del descenso del gradiente se tengan características en una distribución suave en las superficies de error.

### 1.5.2. ¿Qué hace exactamente StandardScaler? Escriba la fórmula matemática de la transformación. (2 pts)

Es una transformación $z = \frac{x-u}{\sigma}$ donde se utiliza la media y la desviación estándar del conjunto de datos para que tengan media cero y desviación estándar de uno.

### 1.5.3. ¿Por qué usamos fit_transform en el conjunto de entrenamiento pero solo transform en el de prueba? (3 pts)

El método transform se le hacen a los datos de prueba para poder ser çomparablesçon datos de entrenamiento, es decir, mientras método de fit_transform calcula los estadísticos necesarios para la normalización de datos, además, evita que se mezclen datos de entrenamiento con los datos de prueba mediante esta distinción.

# 2. Arquitectura del perceptrón

Ahora implementará la estructura básica de un perceptrón para regresión, conectando cada componente con la teoría del capítulo.

## 2.1. Inicialización de parámetros

Implemente la función de inicialización de pesos y sesgo: (6 puntos)

```
def inicializar_parametros(n_caracteristicas):

    # Inicialización aleatoria pequeña para
        pesos y cero para sesgo [cite: 61,
        62]
    w = np.random.randn(n_caracteristicas, 1)
        * 0.01
    b = 0.0
    return w, b
```

### 2.1.1. Complete la función. ¿Por qué es conveniente inicializar los pesos con valores aleatorios pequeños en lugar de ceros? (3 pts)

Se hace con números aleatorios para romper la simetría, de esta manera evitamos que todas las neuronas

aprendan lo mismo, ya que recibirían el mismo gradiente y la misma salida

#### 2.1.2. Según la notación del capítulo, ¿qué forma debe tener el vector w? Verifique con `print(w.shape)` después de llamar a su función. (2 pts)

Se considera el siguiente bloque:

```
n_caracteristicas = X_train_scaled.shape[1]
w, _ = inicializar_parametros(
    n_caracteristicas)
print(w.shape)
```

Debe quedar como (n,1) donde n es el número de características, que en este caso al ser 8 características queda como (8,1), siendo este un vector columna.

#### 2.1.3. ¿Por qué el sesgo b se inicializa típicamente en cero mientras los pesos no? (1 pts)

Se inicia en cero porque su función es desplazar la función de activación; no afecta la ruptura de simetría de los pesos iniciales suma ponderada como si lo hacen los pesos.

### 2.2. Suma ponderada (Forward Propagation)

Implementación de la suma ponderada. Recuerde la ecuación fundamental: $z = w^T x + b$ (10 puntos)

```
def propagacion_adelante(X, w, b):
    # Suma ponderada matricial: z = Xw + b [
        cite: 69, 84]
    y_pred = np.dot(X, w) + b
    return y_pred
```

#### 2.2.1. Complete la función. Explique por qué usamos `X @ w` en lugar de `w^T @ X` cuando X tiene forma (m, n). (3 pts)

Se calcula las predicciones de cada observación al mismo tiempo ya que al trabajar con una matriz x de forma(m, n), la operación del producto punto permite calcular simultáneamente las m predicciones mediante una sola operación de álgebra lineal, resultando en un vector de (m, 1). En caso contrario, se daría un error dimensional de la otra forma.

#### 2.2.2. En un problema de regresión como este, ¿qué función de activación se usa? ¿Por qué? (2 pts)

En regresión simple se usa la función lineal (identidad), ya que necesitamos que el modelo sea capaz de predecir valores continuos sin restricciones de rango, preservando la escala original de los datos de la etiqueta $y$.

#### 2.2.3. Pruebe su función con los primeros 5 ejemplos del conjunto de entrenamiento. Muestre las predicciones iniciales y compárelas con los valores reales. (3 pts)

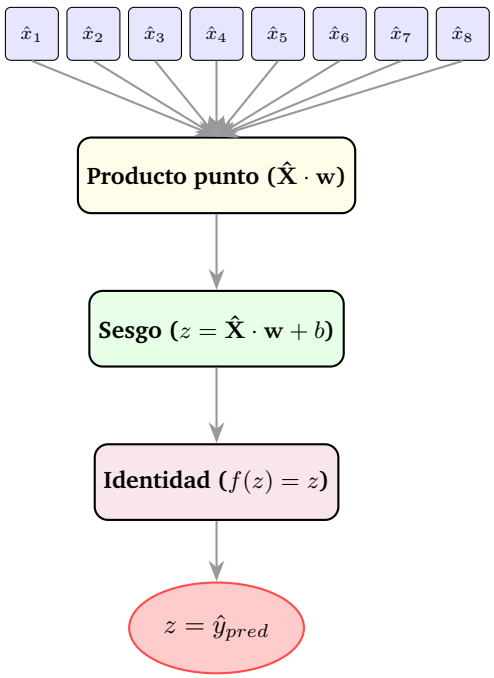
```
# 1. Obtener los primeros 5 ejemplos del
    conjunto de entrenamiento normalizado
X_5 = X_train_scaled[:5]
y_5_real = y_train[:5]

# 2. Inicializar parámetros (sin entrenar aún)

w_ini, b_ini = inicializar_parametros(
    X_train_scaled.shape[1])

# 3. Realizar predicción inicial
y_5_pred = propagacion_adelante(X_5, w_ini,
    b_ini)

# 4. Mostrar resultados
print("Comparativa de Predicciones Iniciales:
    ")
print("-" * 40)
for i in range(5):
    print(f"Ejemplo {i+1}: Real: {y_5_real[i
        ][0]:.4f} | Predicción: {y_5_pred[i
        ][0]:.4f}")
```

Resultado de consola con comparación:

```
   Comparativa de Predicciones Iniciales:
----------------------------------------
Ejemplo 1: Real: 1.0300 | Predicción:-0.0084
Ejemplo 2: Real: 3.8210 | Predicción: 0.0109
Ejemplo 3: Real: 1.7260 | Predicción:-0.0211
Ejemplo 4: Real: 0.9340 | Predicción: 0.0193
Ejemplo 5: Real: 0.9650 | Predicción: 0.0221
```

#### 2.2.4. Dibuje un diagrama mostrando el flujo de datos desde las 8 características hasta la salida `y_pred`. (2 pts)

Diagrama de Flujo: Fordward Propagation



## 2.3. Función de pérdida

8. Para regresión, usaremos el Error Cuadrático Medio (MSE). (9 puntos)

```
def calcular_perdida(y_pred, y_real):
    # Error Cuadrático Medio (MSE) [cite: 97, 99]
    m = y_real.shape[0]
    mse = (1/m) * np.sum((y_pred - y_real)**2)
    return mse
```

### 2.3.1. Complete la función. ¿Por qué elevamos al cuadrado las diferencias en lugar de usar el valor absoluto? (3 pts)

La función cuadrática es continuamente diferenciable en todo su dominio, a diferencia del valor absoluto que presenta una discontinuidad en su derivada en el origen (cero). Esto es esencial para que el algoritmo de descenso del gradiente funcione sin saltos numéricos

### 2.3.2. Calcule la pérdida inicial (con los pesos aleatorios). Guarde este valor para compararlo después del entrenamiento. (2 pts)

```
# 1. Inicializar parámetros
w_inicial, b_inicial = inicializar_parametros(X_train_scaled.shape[1])

# 2. Generar predicciones iniciales (con pesos al azar)
y_pred_inicial = propagacion_adelante(X_train_scaled, w_inicial, b_inicial)

# 3. Calcular la pérdida inicial
perdida_inicial = calcular_perdida(y_pred_inicial, y_train)

print(f"Pérdida inicial (MSE): {perdida_inicial:.4f}")
```

Se obtiene el siguiente resultado por consola:

```
Pérdida inicial (MSE): 5.6237
```

### 2.3.3. ¿Qué ventaja tiene el MSE para el descenso del gradiente comparado con el error absoluto medio (MAE)? (2 pts)

La ventaja principal es que el MSE es derivable en todo su dominio, incluyendo el punto de error cero, a diferencia del MAE, cuya derivada es discontinua en el origen. Además, el gradiente del MSE es proporcional al error, lo que permite una convergencia más suave y precisa (disminuye la velocidad al acercarse al mínimo), mientras que su naturaleza cuadrática penaliza más las desviaciones grandes, dando un ajuste más robusto frente a errores significativos.

### 2.3.4. ¿Qué significa que la función de pérdida se .estabilice"durante el entrenamiento? Relacione con el concepto de convergencia. (2 pts)

Este concepto se relaciona con la convergencia del modelo, y nos dice que el descenso del gradiente ha alcanzado un mínimo, los gradientes son tan pequeños que ya no producen cambios significativos en los pesos.

# 3. Retropropagación y Gradiente Descendente

Esta es la parte central del aprendizaje. Implementará el algoritmo que permite a la red .aprender de sus errores".

## 3.1. Cálculo del gradiente (Teoría)

Para el MSE con activación lineal, derive matemáticamente las expresiones del gradiente. (12 puntos)

### 3.1.1. Partiendo de $L(w) = \frac{1}{m}\sum(\hat{y}_i - y_i)^2$ y $\hat{y} = Xw + b$, demuestre paso a paso que:

$$\frac{\partial L}{\partial w} = \frac{2}{m} X^T(\hat{y} - y)$$

**(4 pts)**

Partimos de la función de costo $L$:

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

Aplicamos la regla de la cadena para derivar respecto al vector de pesos $w$. Sea el error $E_i = (\hat{y}_i - y_i)^2$:

$$\frac{\partial L}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial E_i}{\partial \hat{y}_i}\cdot\frac{\partial \hat{y}_i}{\partial w}$$
$$= \frac{1}{m}\sum_{i=1}^{m} 2(\hat{y}_i - y_i)\cdot\frac{\partial}{\partial w}(x_i w + b)$$

La derivada de la función lineal respecto a $w$ es el vector de características $x_i$:

$$\frac{\partial}{\partial w}(x_i w + b) = x_i$$

Sustituyendo y reordenando:

$$\frac{\partial L}{\partial w} = \frac{2}{m}\sum_{i=1}^{m} x_i(\hat{y}_i - y_i)$$

Por definición de producto matricial, la suma de los productos de las características $(X)$ por los errores es

equivalente a multiplicar por la transpuesta $X^T$:

$$\boxed{\frac{\partial L}{\partial w} = \frac{2}{m} X^T (\hat{y} - y)}$$

#### 3.1.2. Demuestre también que:

$$\frac{\partial L}{\partial b} = \frac{2}{m} \sum (\hat{y}_i - y_i)$$

**(3 pts)**

Nuevamente, aplicamos la regla de la cadena sobre la función de costo $L$:

$$\begin{aligned} \frac{\partial L}{\partial b} &= \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b} (\hat{y}_i - y_i)^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} 2(\hat{y}_i - y_i) \cdot \frac{\partial \hat{y}_i}{\partial b} \end{aligned}$$

Dado que $\hat{y}_i = x_i w + b$, la derivada parcial respecto a $b$ es constante (la derivada de una constante $b$ es 1, y $x_i w$ se trata como constante):

$$\frac{\partial \hat{y}_i}{\partial b} = 1$$

Sustituyendo este resultado en la sumatoria anterior:

$$\begin{aligned} \frac{\partial L}{\partial b} &= \frac{2}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i) \cdot 1 \\ &= \boxed{\frac{2}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)} \end{aligned}$$

Esto indica que el gradiente del sesgo es simplemente el promedio de los errores multiplicado por 2.

#### 3.1.3. ¿Por qué el gradiente s̈eñala la dirección de máxima pendiente ascendente? Explique usando la metáfora de la montaña. (2 pts)

Apunta en esa dirección porque el vector gradiente esta compuesto por las derivadas parciales de la función perdida, siendo estas las que indican hacía que dirección aumenta la función de manera mas rápida. En la metáfora de la montaña, nosotros nos ubicamos cómo ese vector gradiente que al mirar a sus proximidades busca la mayor inclinación (siendo esta la derivada de la función de perdida en el caso del gradiente) buscando una cresta (punto de optimización).

#### 3.1.4. ¿Qué significa el signo negativo en la regla de actualización $w \leftarrow w - \eta \nabla L$? (3 pts)

Siguiendo la lógica de la definición anterior, el gradiente nos lleva a subir la montaña, pero el objetivo real es minimizar la perdida así que obligamos a los datos a reducir el valor de la función de perdida mediante el inverso aditivo de este mismo, buscando así el punto de menor error.

### 3.2. Implementación de gradientes

Implemente la función de cálculo de gradientes. (8 puntos)

```python
def calcular_gradientes(X, y_pred, y_real):
    m = X.shape[0]
    error = y_pred - y_real # [cite: 135]

    # Derivadas parciales según las fórmulas [cite: 113, 114]
    dw = (2/m) * np.dot(X.T, error)
    db = (2/m) * np.sum(error)
    return dw, db
```

#### 3.2.1. Complete la función siguiendo las fórmulas derivadas. (4 pts)

Se puede verificar en el bloque de código anterior.

#### 3.2.2. Verifique que `dw` tenga la misma forma que `w`. ¿Por qué es esto necesario? (2 pts)

Esto es importante para la actualización de parámetros, esta es una operación de álgebra lineal, y de ella sabemos que solo es posible sumar y restar vectores y matrices que poseen las mismas dimensiones o la operación no estará definida.

#### 3.2.3. ¿Qué sucede con los gradientes si todas las predicciones son exactamente iguales a los valores reales? (2 pts)

En este caso tendríamos que los parámetros dejarían de actualizarse, lo que provocaría que el gradiente no transformara el error, ya que este sería el mínimo que puede alcanzar.

### 3.3. Actualización de parámetros

11. Actualización de pesos y sesgo. (10 puntos)

```python
def actualizar_parametros(w, b, dw, db, learning_rate):
    # Regla de actualización: w = w - (nabla) * div(L) [cite: 117, 147, 150]
    w = w - learning_rate * dw
    b = b - learning_rate * db
    return w, b
```

#### 3.3.1. Complete la función. (2 pts)

Se puede verificar en el bloque de código anterior.

#### 3.3.2. ¿Qué es la tasa de aprendizaje (`learning_rate`)? ¿Qué problemas surgen si es muy grande o muy pequeña? (4 pts)

La tasa de aprendizaje es un hiperparámetro que controla el tamaño del paso en la dirección opuesta al gradiente. Si la tasa de aprendizaje es muy pequeña, el entrenamiento será lento pero estable, mientra que si es muy grande, el algoritmo puede divergir u oscilar alrededor del mínimo.

#### 3.3.3. Ejecute una iteración completa: propagación → pérdida → gradientes → actualización. Compare la pérdida antes y después. (4 pts)

Se considero el siguiente bloque de código:

```
# 1. Configuración inicial
learning_rate = 0.01
w, b = inicializar_parametros(X_train_scaled.shape[1]) # Parámetros iniciales

# --- PASO 1: Propagación hacia adelante y pérdida inicial ---
y_pred_antes = propagacion_adelante(X_train_scaled, w, b)
perdida_antes = calcular_perdida(y_pred_antes, y_train)

# --- PASO 2: Cálculo de gradientes ---
dw, db = calcular_gradientes(X_train_scaled, y_pred_antes, y_train)

# --- PASO 3: Actualización de parámetros ---
w, b = actualizar_parametros(w, b, dw, db, learning_rate)

# --- PASO 4: Nueva propagación y pérdida después de la actualización ---
y_pred_despues = propagacion_adelante(X_train_scaled, w, b)
perdida_despues = calcular_perdida(y_pred_despues, y_train)

# --- COMPARACIÓN ---
print(f"Pérdida ANTES de la actualización: {perdida_antes:.6f}")
print(f"Pérdida DESPUÉS de la actualización: {perdida_despues:.6f}")
print(f"¿La pérdida disminuyó?: {'SÍ' if perdida_despues < perdida_antes else 'NO'}")
print(f"Diferencia: {perdida_antes - perdida_despues:.6f}")
```

Comparación:

```
Pérdida ANTES de la actualización: 5.630639
Pérdida DESPUÉS de la actualización: 5.432371
¿La pérdida disminuyó?: SÍ
Diferencia: 0.198269
```

Este disminuyó, esto se debe a que empujamos los pesos a moverse en dirección opuesta al gradiente, donde esta el menor error.

# 4. Entrenamiento Completo

## 4.1. Ciclo de entrenamiento

Implemente el ciclo de entrenamiento completo. (15 puntos)

```
def entrenar_perceptron(X_train, y_train, learning_rate=0.01, epochs=1000):
    n_caracteristicas = X_train.shape[1]
    w, b = inicializar_parametros(n_caracteristicas)
    historial_perdida = []

    for epoca in range(epochs):
        # 1. Propagación adelante # [cite: 177]
        y_pred = propagacion_adelante(X_train, w, b)

        # 2. Pérdida # [cite: 178]
        perdida = calcular_perdida(y_pred, y_train)
        historial_perdida.append(perdida)

        # 3. Gradientes # [cite: 179]
        dw, db = calcular_gradientes(X_train, y_pred, y_train)

        # 4. Actualización # [cite: 180]
        w, b = actualizar_parametros(w, b, dw, db, learning_rate)

        if epoca % 100 == 0:
            print(f'Época {epoca}, Pérdida: {perdida:.4f}') # [cite: 182]

    return w, b, historial_perdida
```

#### 4.1.1. Complete la función integrando todo lo anterior. (5 pts)

Se completó teniendo en cuenta una integración completa.

#### 4.1.2. Entrene el modelo con $lr = 0.01$ y $epochs = 1000$. Grafique el historial de pérdida. (4 pts)

Se implementó cómo se muestra en el bloque:

```
# Ejecución del entrenamiento # [cite: 185]
w_final, b_final, historial = entrenar_perceptron(X_train_scaled, y_train, learning_rate=0.01, epochs=1000)
```

Y se obtuvo la siguiente gráfica:

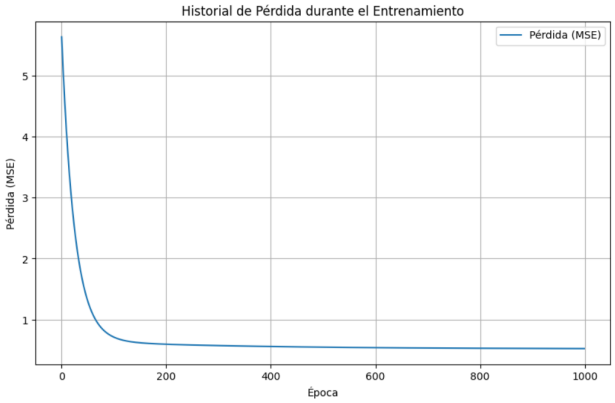



Figura 1: MSE a través de 1000 épocas

#### 4.1.3. ¿El modelo convergió? ¿Cómo lo sabe observando la gráfica? (2 pts)

Se puede ver que a medida que las épocas aumentan, el error disminuye de forma considerable, en la gráfica se puede observar que a partir de cierto punto ($> 500$) el error casi no varía y esto se puede considerar cómo que convergió.

#### 4.1.4. Experimente con tasas: 0.001, 0.01, 0.1, 1.0. Grafique las 4 curvas. ¿Cuál funciona mejor? (4 pts)

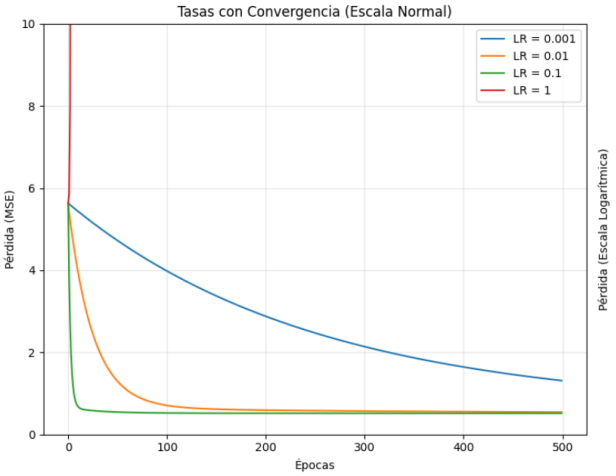



Figura 2: MSE con LR de 0.001, 0.01, 0.1 y 1 a través de 500 épocas

De acuerdo a los resultados a través de las épocas el menor MSE después de 500 épocas las tuvo el LR de 0.01.

### 4.2. Evaluación en conjunto de prueba

Evaluación final. (10 puntos)
Considere el siguiente bloque para los siguientes iterales:

```
# --- EVALUACIÓN FINAL CON EL MEJOR MODELO (LR = 0.01) ---

Entrenamos una última vez con la mejor tasa para asegurar que w y b sean los óptimos
w_opt, b_opt, historial_opt = entrenar_perceptron(X_train_scaled, y_train, learning_rate=0.01, epochs=1000)
```

#### 4.2.1. Calcule las predicciones en `X_test_scaled` usando los parámetros entrenados. (2 pts)

```
# Predicciones en el conjunto de prueba usando los parámetros óptimos
y_pred_test = propagacion_adelante(X_test_scaled, w_opt, b_opt)
```

#### 4.2.2. Calcule el MSE en prueba. ¿Es mayor o menor que el de entrenamiento? ¿Qué indica sobre la generalización? (3 pts)

```
# Cálculo del MSE de prueba
mse_test = calcular_perdida(y_pred_test, y_test)
mse_train_final = historial_opt[-1]

print(f"MSE Final Entrenamiento (LR=0.1): {mse_train_final:.4f}")
print(f"MSE Conjunto de Prueba: {mse_test:.4f}")
```

Respuesta de consola:

```
MSE Final Entrenamiento (LR=0.1): 0.5245
MSE Conjunto de Prueba: 0.5545
```

El MSE de ambos conjuntos son aproximadamente similares, con un MSE ligeramente mayor en la prueba pero no lo suficiente como para ser un sobreajuste considerable. Indica que se tiene unas predicciones funcionales.

#### 4.2.3. Scatter plot: `y_test` vs predicciones. Añada línea $y = x$. (3 pts)

```
# Scatter plot: y_test vs predicciones
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_test, alpha=0.4, color='green', label='Predicciones (LR=0.01)')
# Línea ideal y = x
lims = [np.min([y_test.min(), y_pred_test.min()]), np.max([y_test.max(), y_pred_test.max()])]
plt.plot(lims, lims, 'r--', alpha=0.75, zorder=0, label='Ideal (y=x)')

plt.title('Evaluación Final: Valores Reales vs Predicciones', fontsize=14)
plt.xlabel('Precios Reales (y_test)')
plt.ylabel('Precios Predichos (y_pred_test)')
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

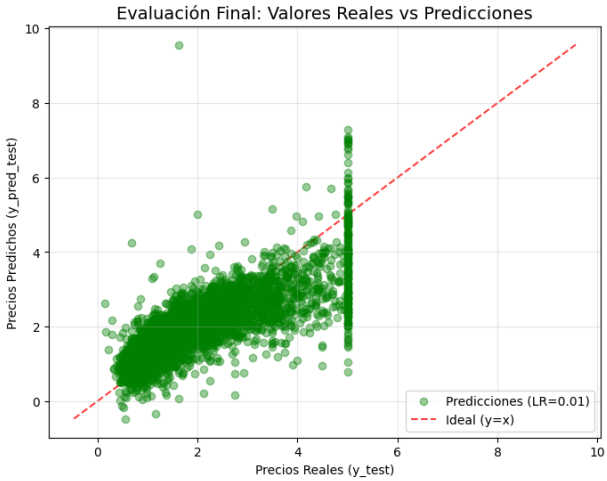



Figura 3: Valores reales vs Predicciones

#### 4.2.4. Calcule $R^2$. Interprete. (2 pts)

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

**Donde** $SS_{res} = \sum(y_{test} - y_{pred})^2$ **y** $SS_{tot} = \sum(y_{test} - \textbf{media}(y_{test}))^2$**.**

```
# Coeficiente de determinación R^2
ss_res = np.sum((y_test - y_pred_test)**2)
ss_tot = np.sum((y_test - np.mean(y_test))**2)

r2 = 1 - (ss_res / ss_tot)

print(f"Coeficiente de determinación R^2
    final: {r2:.4f}")
```

Dando de resultado en consola:

```
    Coeficiente de determinación R^2 final:
        0.5769
```

Esto indica que no solo la métrica dada por el MSE da unos resultados favorables sino que el $R^2$ también se encuentra dentro de un rango que se puede considerar como predicción funcional.

# 5. Reflexión Final (Bonus)

## 5.1. Limitaciones

Limitaciones del perceptrón simple (5 puntos)

#### 5.1.1. ¿Qué tipo de relación entre características y precio puede modelar un perceptrón simple (sin capas ocultas)? (1 pts)

Un perceptrón simple sin capas ocultas únicamente puede modelar relaciones lineales entre las variables de entrada (características) y la salida (precio). Esto se debe a que su arquitectura se basa en una combinación lineal de los pesos y las entradas, por lo que solo es capaz de resolver problemas donde los datos son linealmente separables (ajustables mediante un hiperplano).

#### 5.1.2. Si la relación fuera altamente no lineal, ¿cómo podría extender este modelo? Relacione con perceptrón multicapa. (2 pts)

Para capturar relaciones altamente no lineales, el modelo debe extenderse añadiendo capas ocultas entre la entrada y la salida, las cuales utilizan funciones de activación no lineales. Esta extensión transforma el modelo en un Perceptrón Multicapa (MLP).

#### 5.1.3. El capítulo menciona el Teorema de Aproximación Universal. ¿Qué nos garantiza este teorema sobre las capacidades de una red con una capa oculta? ¿Por qué entonces se usan redes profundas?

El Teorema de Aproximación Universal asegura que una sola capa oculta con suficientes neuronas puede modelar cualquier función continua; sin embargo, se prefieren las redes profundas porque son mucho más eficientes computacionalmente, logrando resolver problemas complejos con una estructura jerárquica y menos neuronas totales que las requeridas por una única capa más grande.

# 6. Link

Aquí fueron desarrollados todos los bloques de código necesarios para la resolución de los ejercicios.
https://github.com/mateolikescats/ML/blob/main/Quiz1.ipynb