

# Capítulo 5

## Teoría de la Aproximación

En numerosas situaciones prácticas dentro de la ciencia y la ingeniería, nos encontramos con conjuntos de datos discretos, representados como pares ordenados  $(x_i, y_i)$ , que provienen de mediciones experimentales o simulaciones. Estos datos a menudo presentan cierta dispersión o ruido inherente al proceso de obtención. El objetivo fundamental de la aproximación por mínimos cuadrados es encontrar una función  $f(x)$ , perteneciente a una familia de funciones predefinida (como líneas rectas, polinomios, exponenciales, etc.) que mejor se ajuste a estos datos observados. La noción de “mejor se ajuste” se formaliza mediante la minimización de la suma de los cuadrados de las diferencias verticales entre los valores observados  $y_i$  y los valores correspondientes predichos por la función  $f(x_i)$ .

Formalmente, para cada punto de dato  $(x_i, y_i)$ , definimos el error o residuo  $e_i$  como la diferencia entre el valor observado  $y_i$  y el valor estimado  $f(x_i)$ , es decir,  $e_i = y_i - f(x_i)$ . La técnica de mínimos cuadrados busca minimizar la función objetivo  $S$ , que se define como la suma de los cuadrados de estos errores para todos los puntos de datos considerados:

$$S = \sum_{i=1}^n (y_i - f(x_i))^2$$

Al minimizar esta suma de cuadrados, buscamos la función  $f(x)$  cuyos valores se encuentren, en promedio y en términos cuadráticos, lo más cerca posible de los valores observados  $y_i$ . El proceso para encontrar esta función óptima implica determinar los parámetros que definen la forma específica de  $f(x)$  de tal manera que la función objetivo  $S$  alcance su valor mínimo.

## Ejemplo

Supongamos que tenemos los siguientes puntos de datos:

$x_i$	$y_i$
1	2
2	3
3	3
4	4
5	5

Nuestro objetivo es encontrar una función lineal de la forma

$$f(x) = a_0 + a_1x$$

que mejor se ajuste a estos puntos utilizando el método de mínimos cuadrados.

Las ecuaciones normales para para regresión lineal son las siguientes

$$n \cdot a_0 + a_1 \left( \sum x_i \right) = \sum y_i$$

$$a_0 \left( \sum x_i \right) + a_1 \left( \sum x_i^2 \right) = \sum x_i y_i$$

Primero calculamos las sumatorias necesarias a partir de nuestros datos (donde  $n = 5$  es el número de puntos):

$$\sum x_i = 1 + 2 + 3 + 4 + 5 = 15$$

$$\sum y_i = 2 + 3 + 3 + 4 + 5 = 17$$

$$\begin{aligned} \sum x_i^2 &= 1^2 + 2^2 + 3^2 + 4^2 + 5^2 \\ &= 1 + 4 + 9 + 16 + 25 = 55 \end{aligned}$$

$$\begin{aligned} \sum x_i y_i &= (1 \times 2) + (2 \times 3) + (3 \times 3) + (4 \times 4) + (5 \times 5) \\ &= 2 + 6 + 9 + 16 + 25 = 58 \end{aligned}$$

Ahora, sustituimos estos valores en las ecuaciones normales:

$$1. 5a_0 + 15a_1 = 17$$

$$2. 15a_0 + 55a_1 = 58$$

Tenemos un sistema de dos ecuaciones lineales con dos incógnitas ( $a_0$  y  $a_1$ ). Podemos resolverlo utilizando métodos como sustitución o eliminación. Usemos el método de eliminación:

Multiplicamos la primera ecuación por  $-3$ .

$$-15a_0 - 45a_1 = -51$$

Ahora sumamos esta ecuación a la segunda ecuación:

$$(15a_0 + 55a_1) + (-15a_0 - 45a_1) = 58 + (-51)$$

$$10a_1 = 7$$

$$a_1 = \frac{7}{10} = 0.7$$

Ahora sustituimos el valor de  $a_1$  en la primera ecuación original para encontrar  $a_0$ :

$$5a_0 + 15(0.7) = 17$$

$$5a_0 + 10.5 = 17$$

$$5a_0 = 17 - 10.5$$

$$5a_0 = 6.5$$

$$a_0 = \frac{6.5}{5} = 1.3$$

Por lo tanto la función lineal que mejor se ajusta a los datos por mínimos cuadrados es:

$$f(x) = 1.3 + 0.7x$$

## 5.1 Implementación en Python

A continuación se presenta el código mediante el cual se resuelve el problema anterior

```
1 # Datos del ejemplo
2 x = np.array([1, 2, 3, 4, 5])
3 y = np.array([2, 3, 3, 4, 5])
4
```

```
5 # Calcular las sumatorias necesarias
6 n = len(x)
7 sum_x = np.sum(x)
8 sum_y = np.sum(y)
9 sum_x_cuadrado = np.sum(x**2)
10 sum_xy = np.sum(x * y)
11
12 # Formar el sistema de ecuaciones
13 A = np.array([[n, sum_x],
14               [sum_x, sum_x_cuadrado]])
15
16 b = np.array([sum_y, sum_xy])
17
18 # Resolver el sistema de ecuaciones lineales
19 coeficientes = np.linalg.solve(A, b)
20 a_0 = coeficientes[0]
21 a_1 = coeficientes[1]
22
23 print(f"Coeficiente a_0 (Intercepto): {a_0:.3f}")
24 print(f"Coeficiente a_1 (Pendiente): {a_1:.3f}")
25
26 # Generar los puntos para graficar el ajuste lineal
27 x_dom = np.linspace(min(x), max(x), 100)
28 y_eval = a_0 + a_1 * x_dom
29
30 # Graficar los datos y la línea de ajuste
31 plt.plot(x_dom, y_eval, label=f"Ajuste lineal: y = {a_0:.3f} + {a_1:.3f}x")
32 plt.scatter(x, y, c='red', zorder=3, label="Datos del Problema")
33 plt.xlabel("Valores de $x$")
34 plt.ylabel("Valores de $y$")
35 plt.title("Aproximación por Mínimos Cuadrados (Regresión Lineal)")
36 plt.legend()
37 plt.grid(linestyle='—')
38
```

```

39 # Guardar la gráfica y mostrar
40 plt.savefig( '.. / Teoria-Aproximacion/Imagenes/Regresion-
    Lineal.pdf')
41 plt.show()

```

La gráfica de la regresión lineal y los datos del problema es la siguiente:

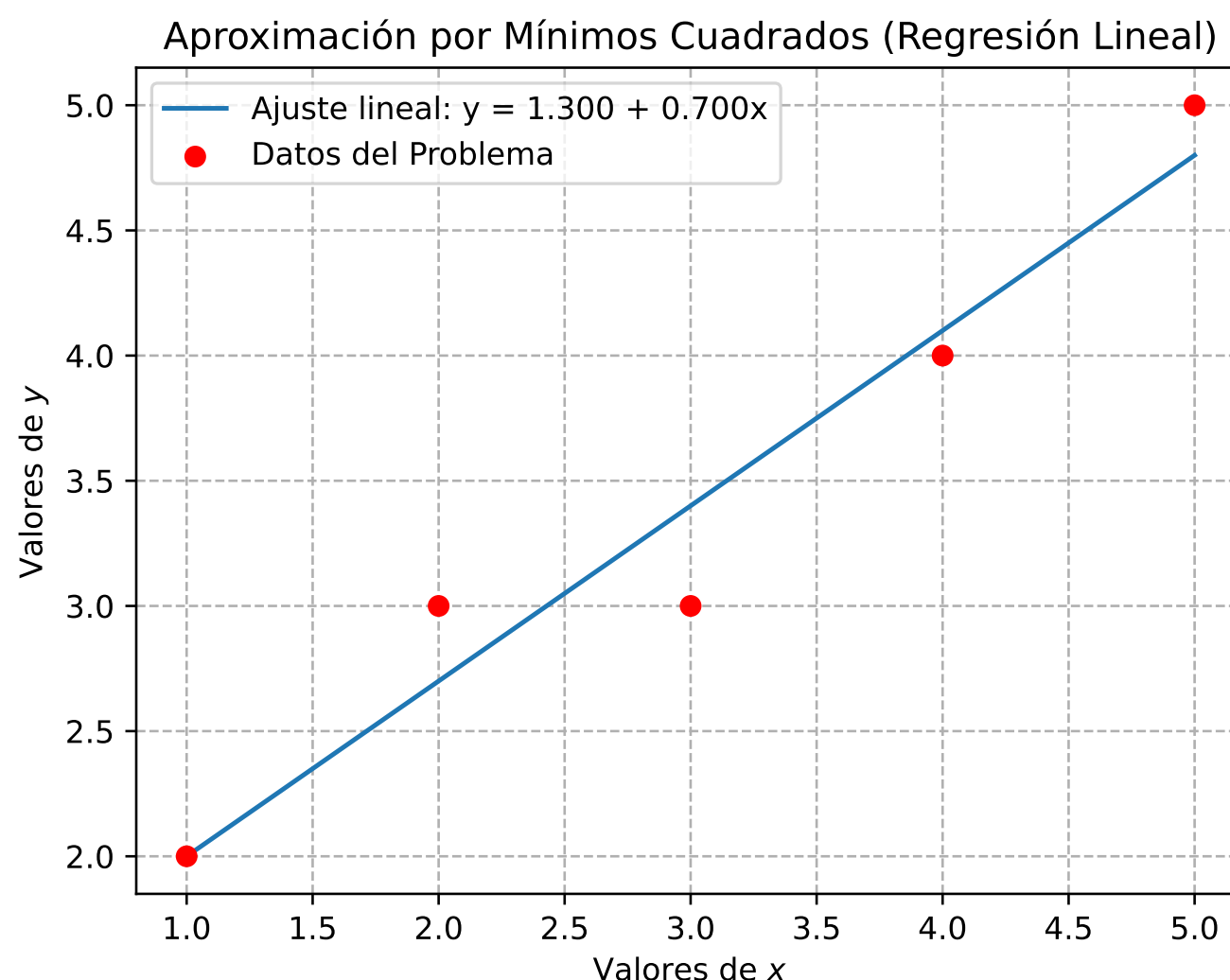


Figura 5.1: Regresión Lineal por Mínimos Cuadrados

## 5.2 Ajuste Lineal Mediante un Perceptron

En esta sección, exploraremos cómo un perceptrón simple puede ser utilizado para realizar un ajuste lineal a un conjunto de datos, similar al método de mínimos cuadrados que discutimos previamente. Si bien la motivación y enfoque difieren, entender esta conexión proporciona una base fundamental para comprender redes neuronales más complejas y sus capacidades de aprendizaje. Utilizaremos el mismo conjunto de datos del ejemplo anterior para comparar los resultados.

Un perceptrón es la forma más básica de procesamiento de una red neuronal que toma una o más entradas, las pondera, suma estas entradas ponderadas, les añade un sesgo (bias), y luego aplica una función de activación para producir una salida. Para realizar una regresión lineal, podemos considerar un perceptrón sin una función de activación no lineal en su salida. En este caso, la salida del perceptrón será una combinación lineal de sus entradas más el sesgo.

Considerando un solo atributo de entrada ( $x$ ) y una salida deseada ( $y$ ), nuestro perceptron para regresión lineal tendrá la siguiente forma

$$\hat{y} = w \cdot x + b$$

Donde

- $\hat{y}$  es la salida predicha del perceptrón (nuestra aproximación a  $y$ )
- $x$  es la entrada
- $w$  es el peso asociado a la entrada  $x$
- $b$  es el sesgo (bias), que actúa como el intercepto en nuestra función lineal.

Nuestro objetivo es encontrar los valores óptimos para el peso  $w$  y el sesgo  $b$  de manera que la salida del perceptrón  $\hat{y}$  se aproxime lo más posible a los valores reales de  $y$  para nuestro conjunto de datos.

### 5.2.1 Aprendizaje del Perceptrón

El perceptrón aprende ajustando sus pesos y sesgos iterativamente basándose en el error entre sus predicciones y los valores reales. Una técnica común para este ajuste es el descenso del gradiente. Definimos una función de costo (similar a la suma de los cuadrados de los errores en mínimos cuadrados) que queremos minimizar. Para la regresión lineal, una función de costo común es el error cuadrático medio (MSE):

$$J(w, b) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 = \frac{1}{n} \sum_{i=1}^n (w \cdot x_i + b - y_i)^2$$

El algoritmo de descenso del gradiente actualiza los pesos y el sesgo en dirección opuesta al gradiente de la función de costo con respecto

a esos parámetros:

$$w = w - \alpha \frac{\partial J}{\partial w}$$

$$b = b - \alpha \frac{\partial J}{\partial b}$$

Donde  $\alpha$  es la tasa de aprendizaje, un hiperparámetro que controla el tamaño del paso en cada iteración.

Calculando las derivadas parciales de la función de costo se tiene:

$$\frac{\partial J}{\partial w} = \frac{2}{n} \sum_{i=1}^n (w \cdot x_i + b - y_i) x_i$$

$$\frac{\partial J}{\partial b} = \frac{2}{n} \sum_{i=1}^n (w \cdot x_i + b - y_i)$$

### Ejemplo 1:

Supongamos que tenemos los siguientes puntos de datos:

$x_i$	$y_i$
1	2
2	3
3	3
4	4
5	5

Nuestro objetivo es encontrar una función lineal de la forma

$$f(x) = w \cdot x + b$$

que mejor se ajuste a estos puntos utilizando el perceptrón simple.

A continuación, implementaremos un perceptrón simple con descenso del gradiente en Python.

```

1 # Datos del ejemplo
2 x = np.array([1, 2, 3, 4, 5])
3 y = np.array([2, 3, 3, 4, 5])
4 n = len(x)
5
6 # Inicialización del peso y el sesgo
7 w = 0.0
8 b = 0.0

```

```
9 tasa_aprendizaje = 0.01
10
11 # Definir el número de iteraciones del aprendizaje
12 epocas = 1000
13
14 # Historial de la función de costo
15 historial = []
16
17 # Entrenamiento mediante el descenso del gradiente
18 for epoca in range(epocas):
19
20     # Calcular las predicciones del perceptron
21     y_ = w*x+b
22
23     # Calcular el costo (MSE)
24     costo = np.mean((y_ - y)**2)
25     historial.append(costo)
26
27     # Calcular los gradientes
28     dw = (2/n) * np.sum((y_ - y) * x)
29     db = (2/n) * np.sum(y_ - y)
30
31     # Actualizar el peso y el sesgo
32     w = w - tasa_aprendizaje * dw
33     b = b - tasa_aprendizaje * db
34
35 # Imprimir el peso y el sesgo
36 print(f"Peso w aprendido: {w:.3f}")
37 print(f"Sesgo b aprendido: {b:.3f}")
38 print(f"La función aproximada por el perceptron es:  $f(x)$   

    = {b:.3f} + {w:.3f}x")
39
40 # Crear el dominio y rango de graficación
41 x_dom = np.linspace(min(x), max(x), 100)
42 y_eval = a_0 + a_1 * x_dom
43
44 # Graficar los datos y la línea de ajuste
```



```

45 plt.plot(x_dom, y_eval, label=f"Ajuste lineal:  $y = \{a_0$ 
     $:.3f\} + \{a_1:.3f\}x$ ")
46 plt.scatter(x, y, c='red', zorder=3, label="Datos del
    Problema")
47 plt.xlabel("Valores de  $x$ ")
48 plt.ylabel("Valores de  $y$ ")
49 plt.title("Aproximación Aprendizaje del Perceptrón")
50 plt.legend()
51 plt.grid(linestyle='—')
52
53 # Guardar la gráfica y mostrar
54 plt.savefig(' ../ Teoria-Aproximacion/Imagenes/Ajuste-
    Lineal-Perceptron.pdf')
55 plt.show()
56
57 # Gráfica de la función de costo durante el
    entrenamiento
58 plt.figure()
59 plt.plot(range(epocas), historial)
60 plt.xlabel('Época')
61 plt.ylabel('Costo (MSE)')
62 plt.title("Evolución del Costo durante el Entrenamiento
    del Perceptrón")
63 plt.grid(linestyle='—')
64
65 # Guardar y mostrar la gráfica
66 plt.savefig(' ../ Teoria-Aproximacion/Imagenes/Evolucion-
    Costo.pdf')

```

El resultado es igual que cuando resolvimos usando el método de mínimos cuadrados. A continuación se muestra la gráfica de la evolución del aprendizaje del perceptrón.

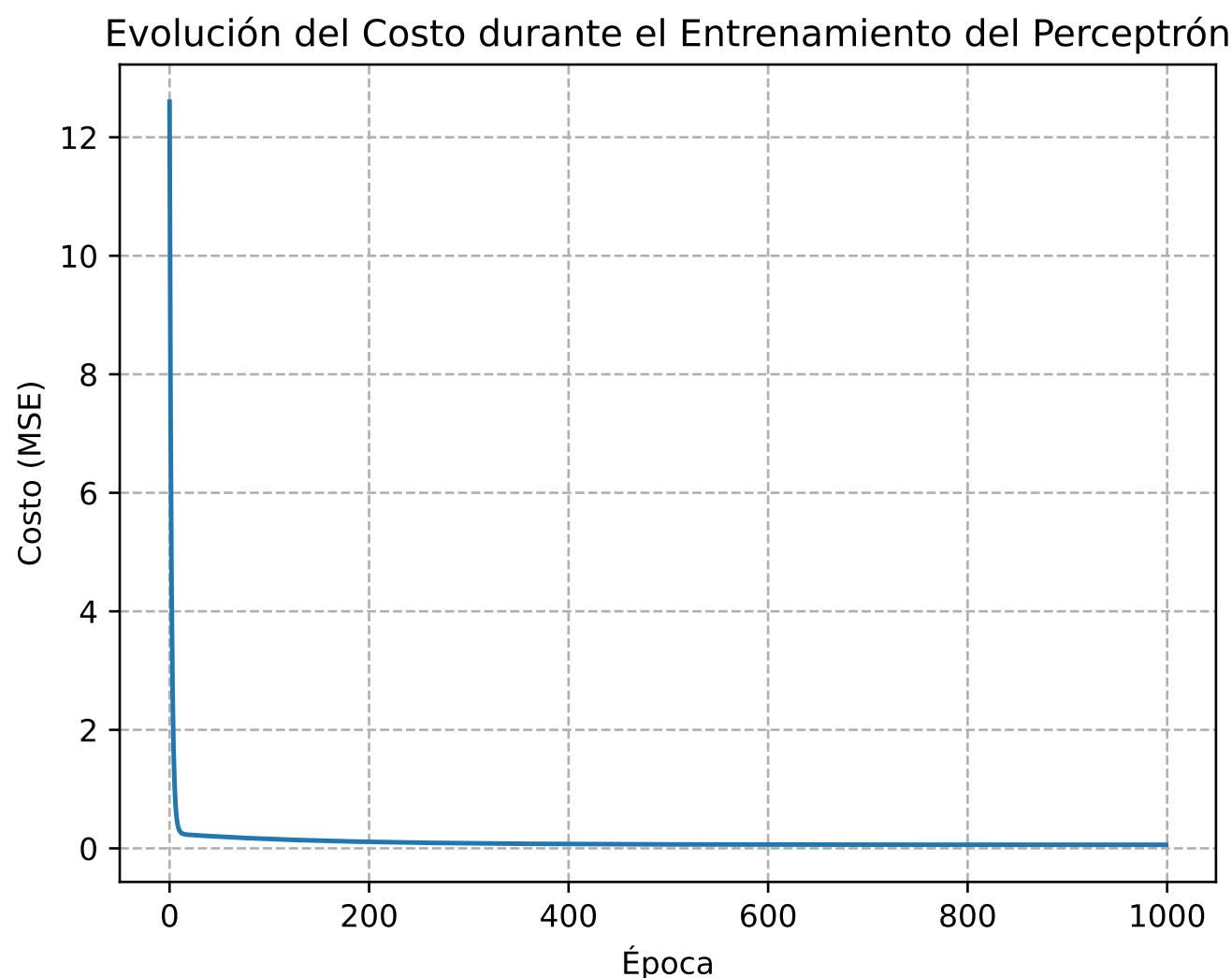


Figura 5.2: Entrenamiento de un perceptrón para aproximar una función lineal

## 5.3 Ajuste Cuadrático Mediante un Perceptron con Expansión de Características

En esta sección, extenderemos el concepto de perceptrón para aproximar una función cuadrática. Para lograr esto con perceptrón lineal, emplearemos la técnica de expansión de características. Transformaremos nuestros datos de entrada originales para que el perceptrón pueda aprender una relación no lineal en el espacio original, pero que sea lineal en el espacio de características expandido.

### 5.3.1 Generación de datos Cuadráticos con Perturbaciones

Comenzaremos generando un conjunto de datos que siga aproximadamente una función cuadrática.

Utilizaremos como ejemplo la función

$$f(x) = -x^2 + 4.75x - 5.08$$

y añadiremos pequeñas perturbaciones aleatorias para simular datos del mundo real con cierto nivel de ruido. Generaremos 12 puntos distribuidos uniformemente en el intervalo  $[0, 5]$ , tal como veremos en el siguiente código de Python.

```

1 # Función cuadrática generadora
2 def funcion_cuadratica(x):
3     return -x**2 + 4.75*x - 5.08
4
5 # Generar 12 puntos en el intervalo [0, 5]
6 x = np.linspace(0, 5, 12)
7 y_real = funcion_cuadratica(x)
8
9 # Lista de las perturbaciones aleatorias
10 ruido = 0.5 * np.random.rand(len(x))
11 y = y_real + ruido
12
13 # Visualizar los datos generados
14 plt.plot(x, y_real, label='Función Cuadrática Generadora')
15 plt.scatter(x, y, c='red', zorder=3, label='Datos Generados')
16 plt.xlabel('Valores de $x$')
17 plt.ylabel('Valores de $y$')
18 plt.title('Datos Generados para la Aproximación Cuadrática')
19 plt.legend()
20 plt.grid(linestyle='—')
21
22 # Guardar y mostrar la gráfica
23 plt.savefig('../Teoria-Aproximacion/Imagenes/Datos-Generados-Funcion-Cuadratica.pdf')
24 plt.show()

```

La gráfica generada es la siguiente

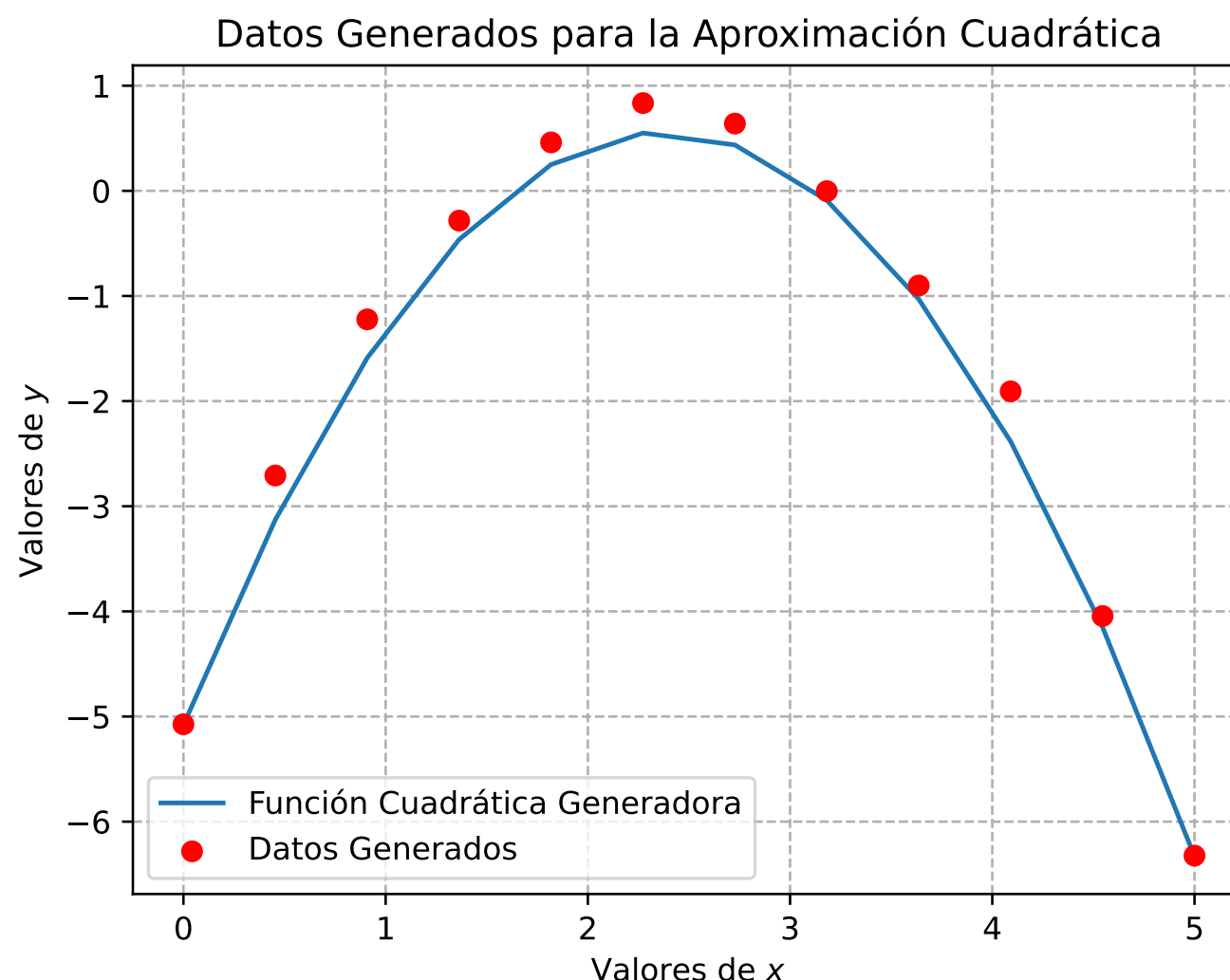


Figura 5.3: Gráfico de los Datos Generados

### 5.3.2 Aproximación Cuadrática con un Perceptron y Expansión de Características

Un perceptrón estándar con una sola entrada solo puede aprender relaciones lineales. Para que pueda aproximar una función cuadrática de la forma

$$\hat{y} = w_2 \cdot x^2 + w_1 \cdot x + b$$

necesitamos expandir nuestras características de entrada. En lugar de alimentar directamente el valor de  $x$  al perceptrón, crearemos un nuevo vector de características para cada punto de datos ( $x_i$ ):

$$\mathbf{x}'_i = [x_i^2 \ x_i \ 1]$$

Aquí, hemos creados dos nuevas “entradas” para nuestro perceptrón:  $x_i^2$  y  $x_i$  y el término constante “1” se multiplicará por el sesgo. Ahora, nuestro perceptrón lineal operará en este espacio de características expandido. La salida del perceptrón será:

$$\hat{y}_i = w_2 (x_i^2) + w_1 (x_i) + b (1)$$

Donde:

- $\hat{y}_i$  es la salida predicha para el punto  $x_i$
- $w_2$  es el peso asociado a la característica  $x_i^2$
- $w_1$  es el peso asociado a la característica  $x_i$
- $b$  es el sesgo (peso asociado a la característica constante 1)

Ahora, el problema se convierte en encontrar los valores óptimos para los pesos  $w_2, w_1$  y el sesgo  $b$  que minimicen una función de costo, como el Error Cuadrático Medio (MSE), entre las predicciones  $\hat{y}_i$  y los valores reales  $y_i$ .

### 5.3.2.1 La función de Costo ( $MSE$ )

La función de costo que utilizaremos es el Error Cuadrático Medio, definida como:

$$J(w_2, w_1, b) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 = \frac{1}{n} \sum_{i=1}^n (w_2 \cdot x_i^2 + w_1 \cdot x_i + b - y_i)^2$$

Nuestro objetivo es encontrar los valores de  $w_2, w_1$ , y  $b$  que minimicen esta función de costo.

### 5.3.2.2 Aprendizaje Mediante Descenso del Gradiente

Al igual que en el caso del ajuste lineal, utilizaremos el algoritmo de descenso del gradiente para encontrar los pesos y el sesgo óptimos. Necesitamos calcular las derivadas parciales de la función de costo con respecto a cada parámetro:

$$\begin{aligned} \frac{\partial J}{\partial w_2} &= \frac{2}{n} \sum_{i=1}^n (w_2 \cdot x_i^2 + w_1 \cdot x_i + b - y_i) \cdot x_i^2 \\ \frac{\partial J}{\partial w_1} &= \frac{2}{n} \sum_{i=1}^n (w_2 \cdot x_i^2 + w_1 \cdot x_i + b - y_i) \cdot x_i \\ \frac{\partial J}{\partial b} &= \frac{2}{n} \sum_{i=1}^n (w_2 \cdot x_i^2 + w_1 \cdot x_i + b - y_i) \end{aligned}$$

Luego actualizaremos los pesos y el sesgo iterativamente:

$$w_2 = w_2 - \alpha \frac{\partial J}{\partial w_2}$$
$$w_1 = w_1 - \alpha \frac{\partial J}{\partial w_1}$$
$$b = b - \alpha \frac{\partial J}{\partial b}$$

donde  $\alpha$  es la tasa de aprendizaje.

### 5.3.2.3 Implementación en Python

A continuación, implementaremos el perceptrón con expansión de características y el algoritmo de descenso del gradiente en Python para aproximar la función cuadrática.

```
1 # Función cuadrática generadora
2 def funcion_cuadratica(x):
3     return -x**2 + 4.75*x - 5.08
4
5 # Generar 12 puntos en el intervalo [0, 5]
6 x = np.linspace(0, 5, 12)
7 y_real = funcion_cuadratica(x)
8
9 # Lista de las perturbaciones aleatorias
10 ruido = 0.5 * np.random.rand(len(x))
11 y = y_real + ruido
12
13 # Inializamos los pesos y el sesgo
14 w2 = 0.0
15 w1 = 0.0
16 b = 0.0
17 alfa = 0.0001
18 epocas = 200000
19 historial_costo = []
20
21 # Entrenamos el perceptron mediante descenso del
    gradinet
22 for epoca in range(epocas):
```

```

23
24 # Calcular las predicciones del perceptrón con las
    características expandidas
25 y_ = w2 * x**2 + w1 * x + b
26
27 # Calcular el costo (MSE)
28 costo = np.mean((y_ - y)**2)
29 historial_costo.append(costo)
30
31 if (costo < 0.01):
32     break
33
34 # Calcular los gradientes
35 dw2 = (2/n) * np.sum((y_ - y) * x**2)
36 dw1 = (2/n) * np.sum((y_ - y) * x)
37 db = (2/n) * np.sum(y_ - y)
38
39 # Actualizamos los pesos y el sesgo
40 w2 = w2 - alfa * dw2
41 w1 = w1 - alfa * dw1
42 b = b - alfa * db
43
44 # Imprimir el costo cada 10000 épocas para
    monitorear el progreso (opcional)
45 if (epoca + 1) % 20000 == 0:
46     print(f'Época {epoca + 1}, Costo: {costo:.6f}')
47
48 # Imprimir los resultados
49 print(f"Peso w2 aprendido: {w2:.3f}")
50 print(f"Peso w1 aprendido: {w1:.3f}")
51 print(f"Sesgo aprendido: {b:.3f}")
52 print(f"Función Cuadrática Aproximada:  $f(x) = {w2:.3f}x^2 + {w1:.3f}x + {b:.3f}$ ")
53
54 # Generar los puntos para la curva de ajuste
55 x_dom = np.linspace(min(x), max(x), 100)
56 y_eval = w2 * x_dom**2 + w1 * x_dom + b

```



```
57
58 # Graficar los datos de aprendizaje y la aproximación
59 plt.plot(x_dom, y_eval, label='Función Aproximada')
60 plt.scatter(x, y, c='red', zorder=3, label='Datos de
    Entrenamiento')
61 plt.xlabel('Valores de  $x$ ')
62 plt.ylabel('Valores de  $y$ ')
63 plt.title('Aproximación Cuadrática Mediante un
    Perceptrón')
64 plt.legend()
65 plt.grid(linestyle='—')
66
67 # Guardar la gráfica y mostrar
68 plt.savefig('../Teoria-Aproximacion/Imagenes/
    Aproximacion-Funcion-Cuadratica.pdf')
69 plt.show()
70
71 # Graficar la función de costo durante el entrenamiento
72 plt.figure()
73 plt.plot(range(epocas), historial_costo)
74 plt.xlabel('Época')
75 plt.ylabel('Costo (MSE)')
76 plt.title(f'Evolución del Costo (MSE), LR={alfa}')
77 plt.grid(linestyle='—')
78
79 # Guardar la gráfica y mostrar
80 plt.savefig('../Teoria-Aproximacion/Imagenes/Evolucion-
    Aprendizaje-Aproximacion-Funcion-Cuadratica.pdf')
81 plt.show()
```

La gráfica de la evolución del aprendizaje se muestra en la siguiente figura:



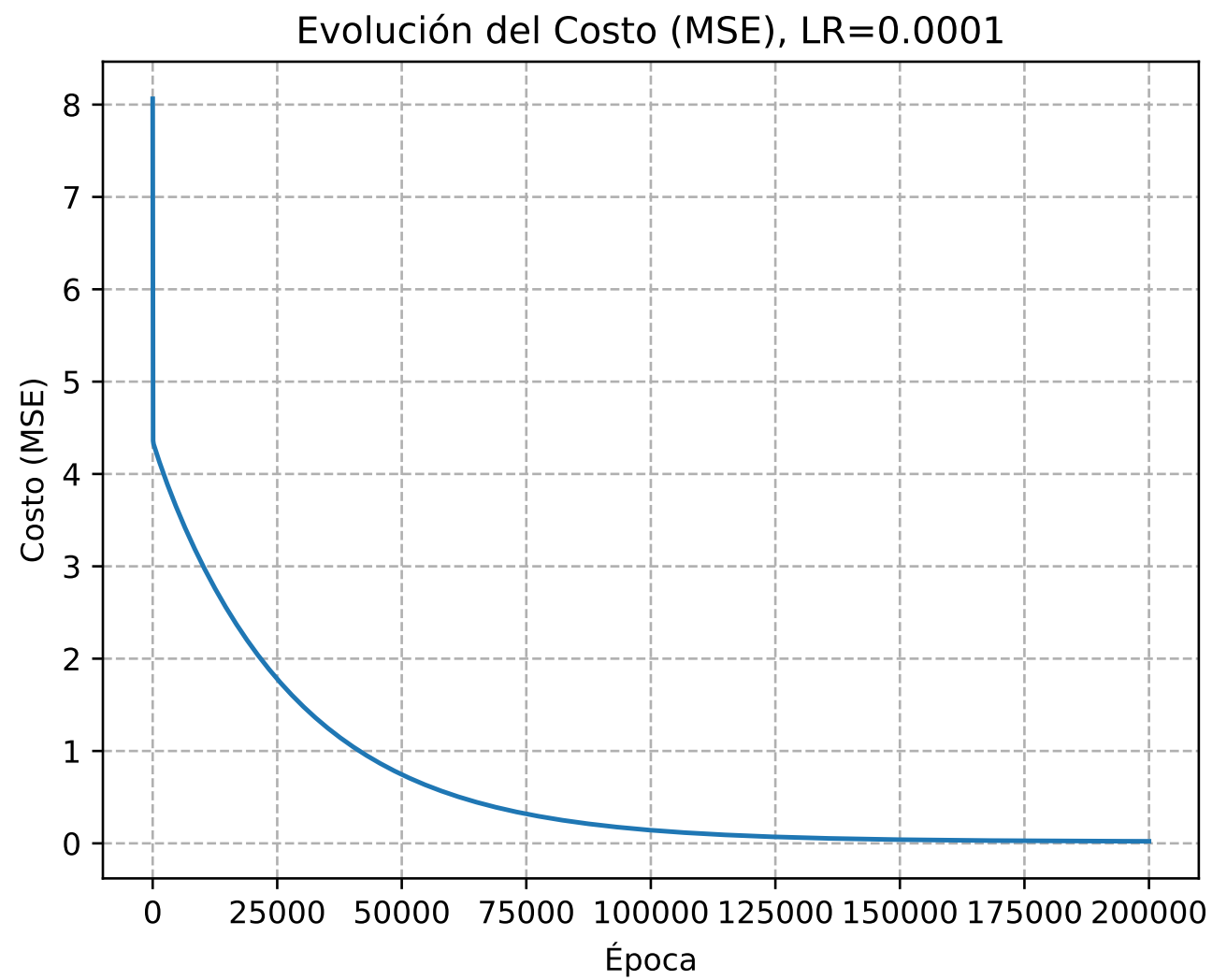


Figura 5.4: Evolución del Aprendizaje de la Aproximación de la Función Cuadrática

y el ajuste se presenta en la siguiente figura:

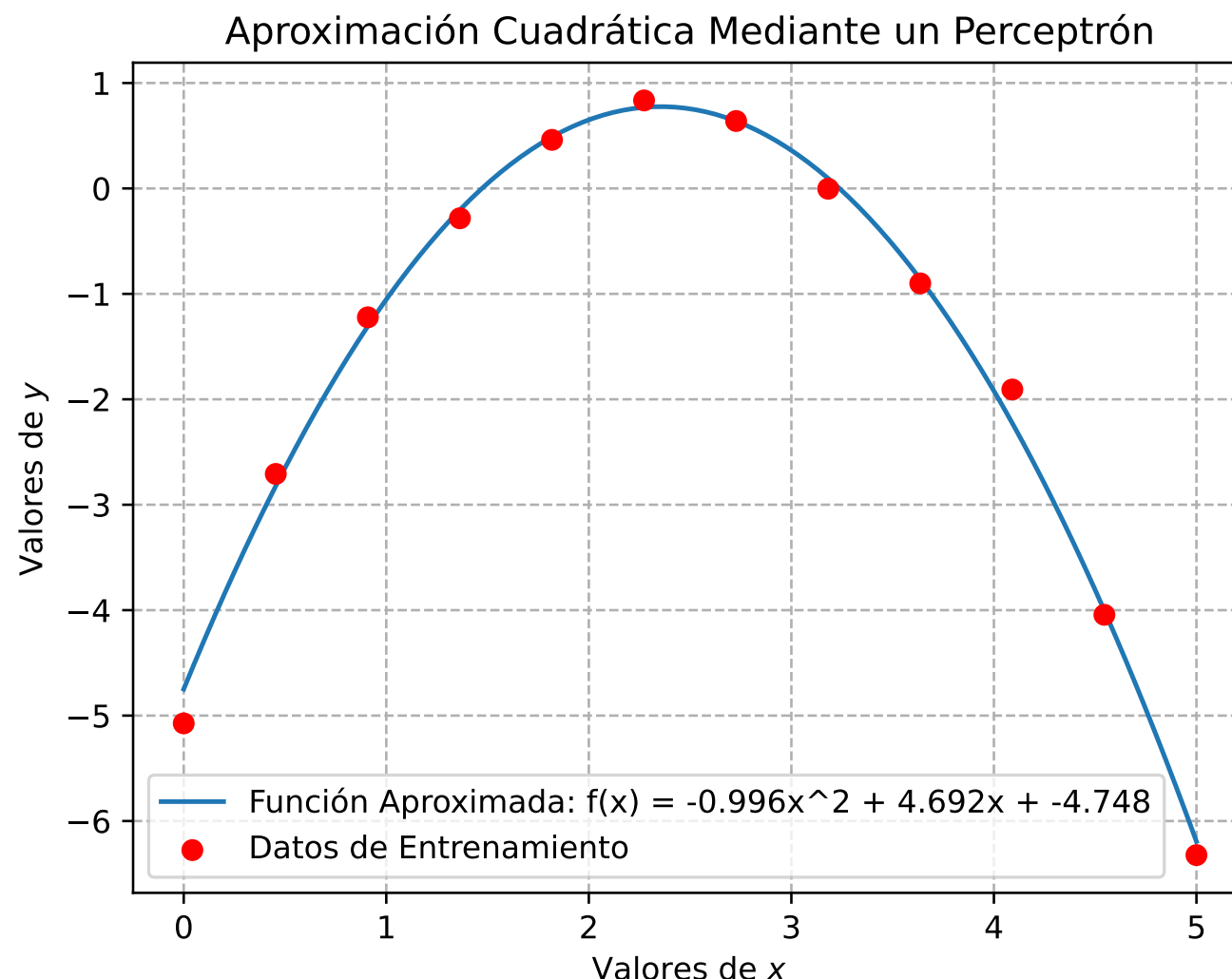


Figura 5.5: Ajuste de la función cuadrática

En particular este método se vuelve más costoso, porque la tasa de aprendizaje es muy pequeña y la cantidad de épocas es muy grande.

## 5.4 Aproximación Polinomial

Cómo vimos en las secciones anteriores, existen diversos métodos para ajustar un conjunto de puntos (nodos) en  $\mathbb{R}^2$  a funciones polinómicas, por ejemplo la interpolación o la regresión polinómica. En esta sección se quiere crear un método a través de la implementación de redes neuronales para encontrar los parámetros de la función polinómica.

Sea una función polinómica definida por

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

que también se puede representar mediante la serie

$$P_n(x) = \sum_{k=0}^n a_k x^k$$

El objetivo del método es encontrar el conjunto de coeficientes del polinomio.