INSTITUTO POLITÉCTICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

UNIDAD DE APRENDIZAJE:

MACHINE LEARNING

PRÁCTICA 1: Regresión Lineal

INTEGRANTES:

Hernández Hernández Roberto Isaac

Gonzalez Llamosas Noe Ramses

PROFESOR:

Ortiz Castillo Marco Antonio

FECHA DE ENTREGA: 14/10/2023

CONTENIDO

1. PRÁCTICA: REGRESIÓN LINEAL

1.1. INTRODUCCIÓN

1.1.1. Regresión Lineal Simple

La regresión lineal es una técnica de análisis de datos que predice el valor de datos desconocidos mediante el uso de otro valor de datos relacionado y conocido. Modela matemáticamente la variable desconocida o dependiente y la variable conocida o independiente como una ecuación lineal. Los modelos de regresión lineal son relativamente simples y proporcionan una fórmula matemática fácil de interpretar para generar predicciones. La regresión lineal es una técnica estadística establecida y se aplica fácilmente al software y a la computación. Las empresas lo utilizan para convertir datos sin procesar de manera confiable y predecible en inteligencia empresarial y conocimiento práctico. Los científicos de muchos campos, incluidas la biología y las ciencias del comportamiento, ambientales y sociales, utilizan la regresión lineal para realizar análisis de datos preliminares y predecir tendencias futuras. Muchos métodos de ciencia de datos, como el machine learning y la inteligencia artificial, utilizan la regresión lineal para resolver problemas complejos [1].

En el machine learning, los programas de computación denominados algoritmos analizan grandes conjuntos de datos y trabajan hacia atrás a partir de esos datos para calcular la ecuación de regresión lineal. Los científicos de datos primero entrenan el algoritmo en conjuntos de datos conocidos o etiquetados y, a continuación, utilizan el algoritmo para predecir valores desconocidos [1]. La regresión lineal simple se define mediante la ecuación 1:

$$Y_{obt} = a * X + b \dots (1)$$

Donde a y b son dos constantes desconocidas que representan la pendiente de regresión y se pueden determinar mediante las ecuaciones 2 y 3.

$$a = \frac{m * \sum(X - Y_d) - \sum(X) * \sum(Y_d)}{m * \sum(X^2) - \sum(X)^2} \dots (2)$$

$$b = \frac{\sum(Y_d) - a - \sum(X)}{m} \dots (3)$$

Donde:

$m =$ Numero de pruebas realizadas

$X =$ Vector de características

$Yd =$ Vector deseado

1.1.2. Regresión Lineal mediante actualización de pesos

Se refiere a un enfoque iterativo para ajustar los coeficientes (pesos) del modelo de regresión lineal, comúnmente asociado con el método de descenso de gradiente. En lugar de calcular directamente los coeficientes a partir de fórmulas cerradas (como en la regresión lineal clásica), este método ajusta los pesos gradualmente a lo largo de múltiples iteraciones, lo que lo hace útil para grandes conjuntos de datos o cuando no se puede calcular la solución exacta de manera eficiente [2]. Considerando de igual forma la ecuación 1, El objetivo es encontrar los valores óptimos de a y b minimizando una función de costo, típicamente el error cuadrático medio (MSE), que se muestra en la ecuación 3:

$$MSE = \frac{1}{2m}\sum_{i=1}^{m}(Y_d - Y_{obt})^2$$

El descenso de gradiente es una técnica iterativa para minimizar la función de costo ajustando los pesos a y b poco a poco en la dirección del gradiente negativo (la dirección de la mayor disminución en el error), para esto se debe realizar una inicializan de los pesos con valores aleatorios o con ceros y calcular el gradiente de la función de costo con respecto a los pesos [2]. Las ecuaciones 4 y 5 determinan a y b mediante este método:

$$a = a - \frac{\eta}{m} * \sum_{i=0}^{m}(Y_{obt} - Y_d)X \ldots (4)$$

$$b = b - \frac{\eta}{m} * \sum_{i=0}^{m}(Y_{obt} - Y_d) \ldots (5)$$

Donde:

$\eta =$ Constante llamada Learning Rate

Se repiten los pasos de cálculo del gradiente y actualización de pesos hasta que los cambios en la función de costo sean lo suficientemente pequeños o hasta alcanzar un número determinado de iteraciones, a esto se le conoce como épocas [2].

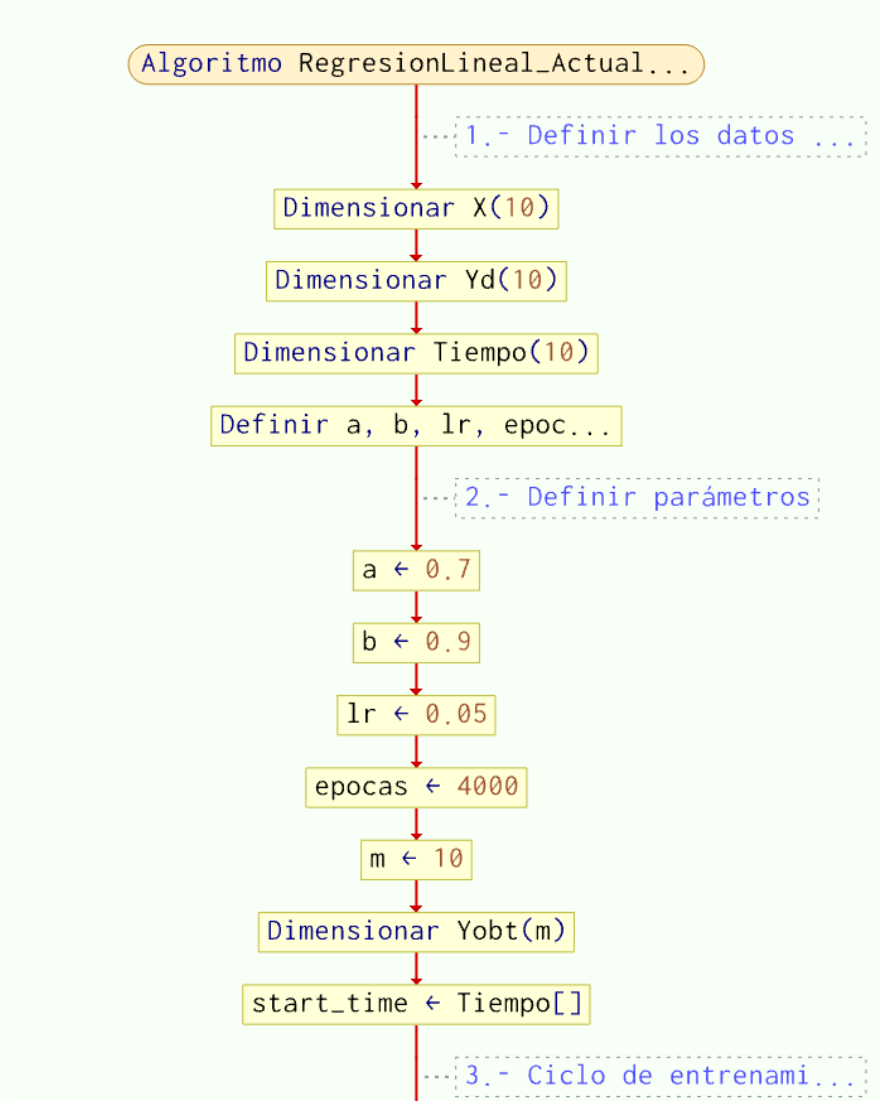
1.2. DESARROLLO

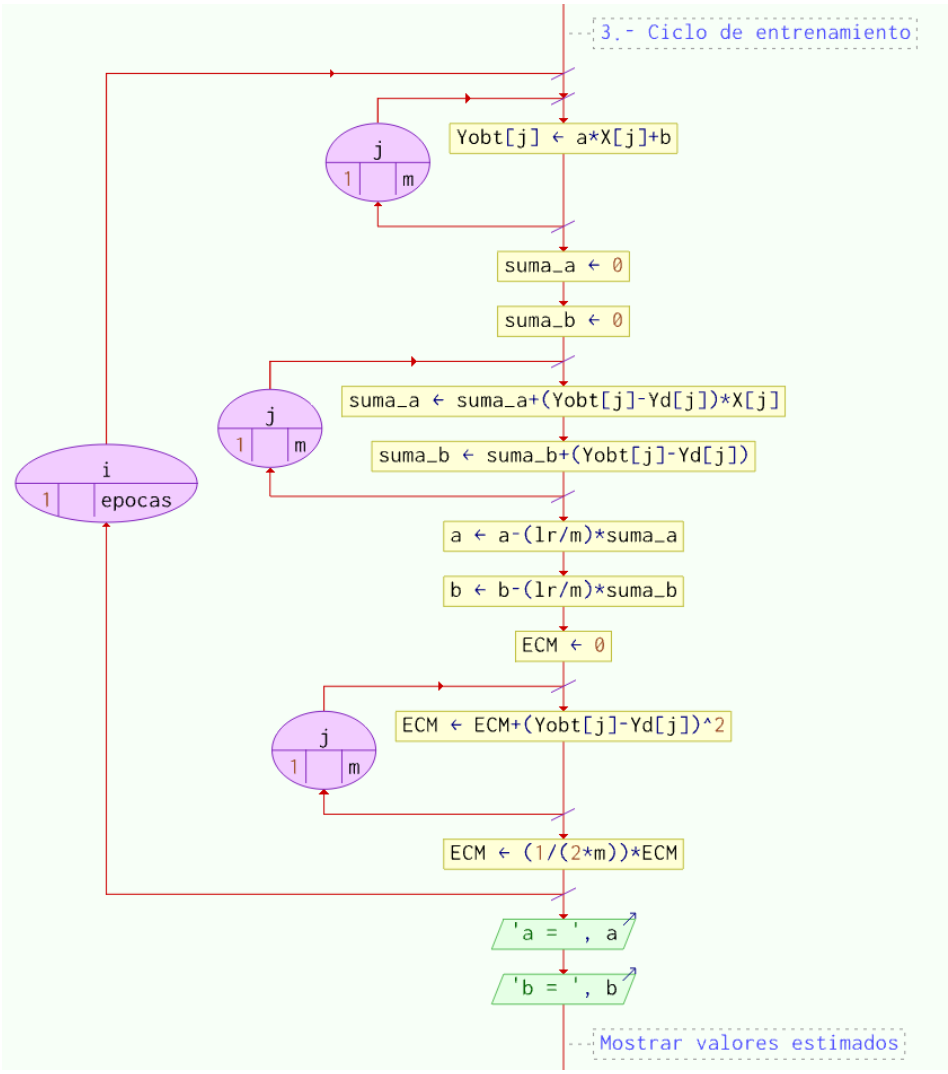
1.2.1. Ejercicio A

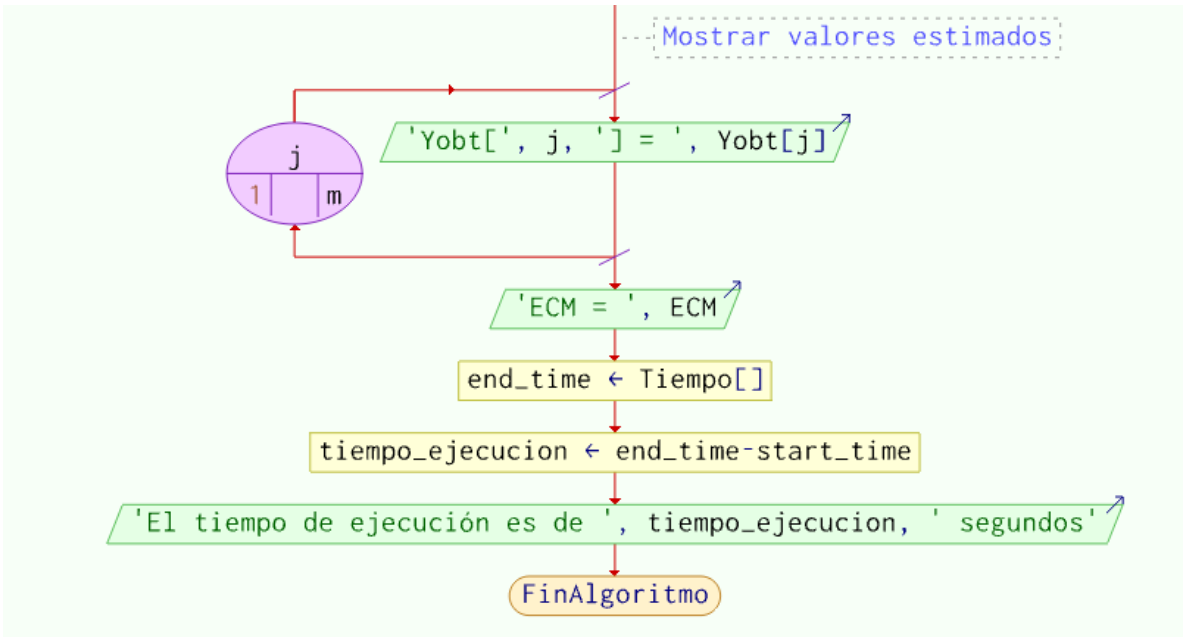
Considerando las ecuaciones 2 y 3 para determinar el valor de a y b mediante la regresión lineal simple, implementamos un código para comparar dichos resultados obtenidos mediante la regresión lineal vía actualización de pesos, cambiando 3 valores diferentes en su Learning Rate y midiendo el tiempo de ejecución de cada uno de estos.

a) Diagrama de flujo

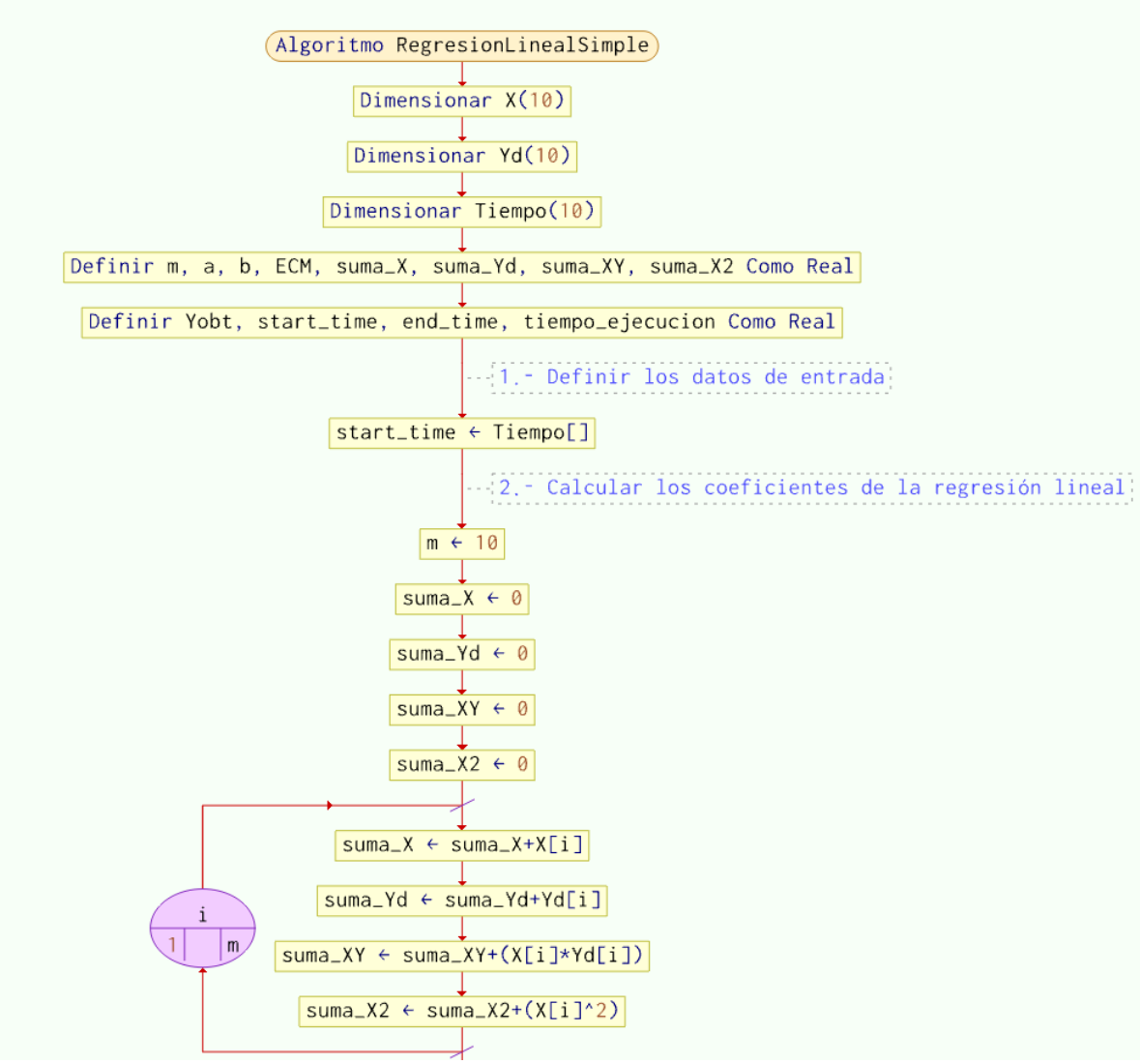
A continuación, se muestra el diagrama de flujo del código implementado para la **regresión lineal vía actualización de pesos.**

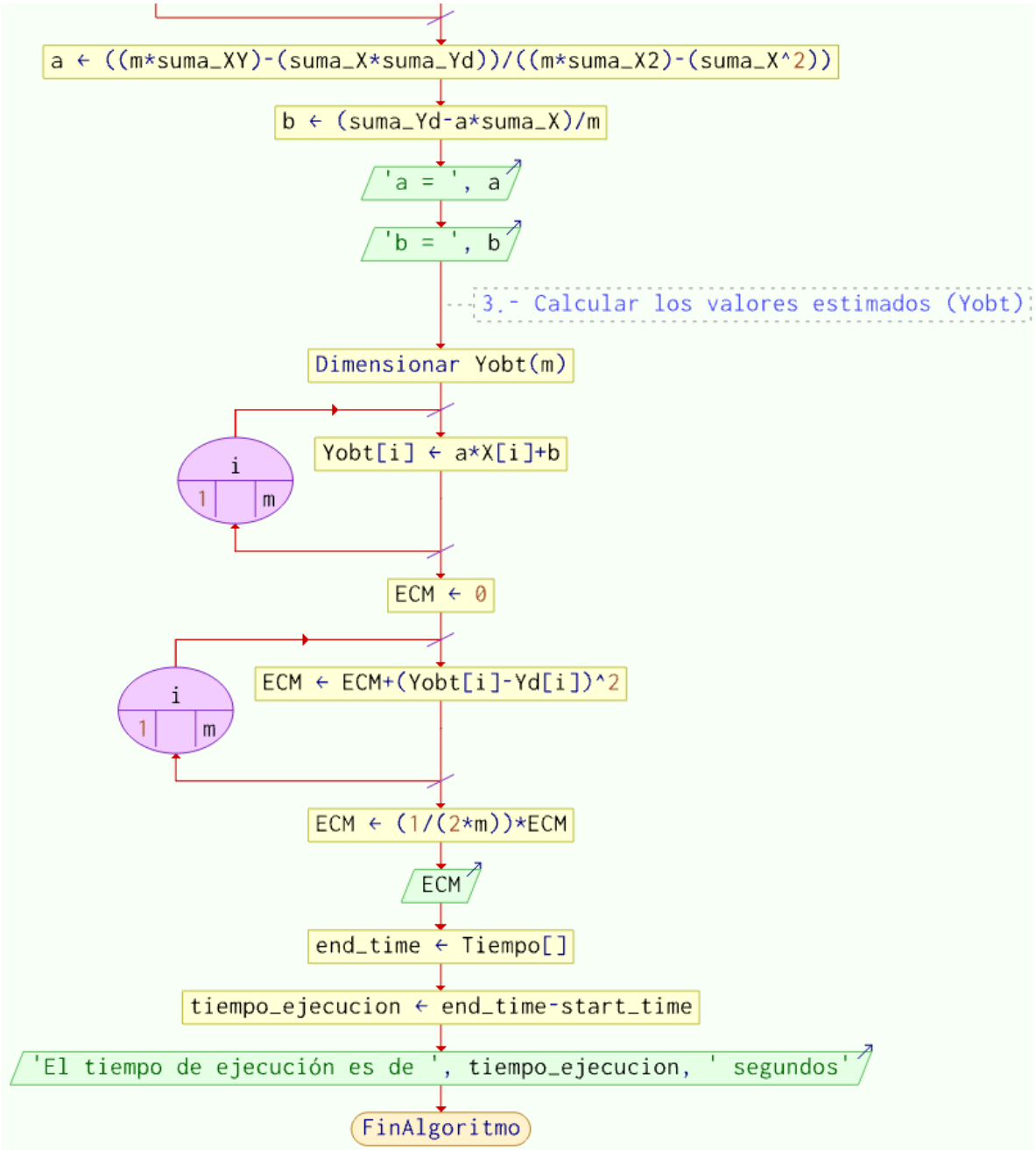



3.- Ciclo de entrenamiento
j
1
m
Yobt[j] ← a*X[j]+b
suma_a ← 0
suma_b ← 0
j
1
m
suma_a ← suma_a+(Yobt[j]-Yd[j])*X[j]
suma_b ← suma_b+(Yobt[j]-Yd[j])
i
1
epocas
a ← a-(lr/m)*suma_a
b ← b-(lr/m)*suma_b
ECM ← 0
j
1
m
ECM ← ECM+(Yobt[j]-Yd[j])^2
ECM ← (1/(2*m))*ECM
'a = ', a
'b = ', b
Mostrar valores estimados


Mostrar valores estimados
j
1
m
'Yobt[', j, '] = ', Yobt[j]
'ECM = ', ECM
end_time ← Tiempo[]
tiempo_ejecucion ← end_time-start_time
'El tiempo de ejecución es de ', tiempo_ejecucion, ' segundos'
FinAlgoritmo

A continuación, se muestra el diagrama de flujo del código implementado para la **regresión lineal simple.**




a ← ((m*suma_XY)-(suma_X*suma_Yd))/((m*suma_X2)-(suma_X^2))
b ← (suma_Yd-a*suma_X)/m
'a = ', a
'b = ', b
3.- Calcular los valores estimados (Yobt)
Dimensionar Yobt(m)
i
1
m
Yobt[i] ← a*X[i]+b
ECM ← 0
i
1
m
ECM ← ECM+(Yobt[i]-Yd[i])^2
ECM ← (1/(2*m))*ECM
ECM
end_time ← Tiempo[]
tiempo_ejecucion ← end_time-start_time
'El tiempo de ejecución es de ', tiempo_ejecucion, ' segundos'
FinAlgoritmo

b) Códigos

```
# Regresion lineal vía actualización de pesos
import numpy as np
import matplotlib.pyplot as plt
import time

# 1.- Definir los datos de entrada
X = np.array([1,2,3,4,5,6,7,8,9,10])
Yd = np.array([0.8,2.95,2.3,3.6,5.2,5.3,6.1,5.9,7.6,9])

# 2.- Definir parametros
a = 0.7
b = 0.9
lr = 0.05
epocas = 4000
m = len(Yd)
Yobt = np.zeros(m)

start_time = time.time()

for i in range(epocas):
   # 3.- Calculamos Yobt
   Yobt = a * X + b
   a -= (lr / m) * np.sum((Yobt - Yd) * X)
   b -= (lr / m) * np.sum((Yobt - Yd))
   ECM = (1 / (2 * m)) * np.sum(Yobt - Yd) **2

print(f"a = { a }")
print(f"b = { b }")

end_time = time.time()
print(f"El tiempo de ejecución es de {end_time - start_time} segundos")

# 4.- Grafica de las funciones
plt.scatter(X, Yd, color='blue', label='Datos originales')
plt.plot(X, Yobt, color='red', label='Recta de regresión')

plt.xlabel('X')
plt.ylabel('Yd')
plt.title('Regresión Lineal Vía Actualización de Pesos')
plt.show()
```

```python
# Regresion lineal simple
import numpy as np
import matplotlib.pyplot as plt
import time

# 1.- Definir los datos de entrada
X = np.array([1,2,3,4,5,6,7,8,9,10])
Yd = np.array([0.8,2.95,2.3,3.6,5.2,5.3,6.1,5.9,7.6,9])

start_time = time.time()

# 2.- Calcular los coeficientes de la regresión lineal
m = len(Yd)
a = ((m * np.sum(X * Yd)) - (np.sum(X) * np.sum(Yd))) / ((m * np.sum(X **
2)) - (np.sum(X) ** 2))
b = (np.sum(Yd) - a * np.sum(X)) / m
print(f"a = {a}")
print(f"b = {b}")

# 3.- Calcular los valores estimados (Yobt)
Yobt = a * X + b

end_time = time.time()
print(f"El tiempo de ejecución es de {end_time - start_time} segundos")

# 4.- Grafica de las funciones
plt.scatter(X, Yd, color='blue', label='Datos originales')
plt.plot(X, Yobt, color='red', label='Recta de regresión')

plt.xlabel('X')
plt.ylabel('Yd')
plt.title('Regresión Lineal Simple')

plt.show()
```

c) Funcionamiento

En la Figura 1.1 se muestra el funcionamiento y tiempo de ejecución del código vía actualización de pesos mediante un Learning Rate de **0.05**.

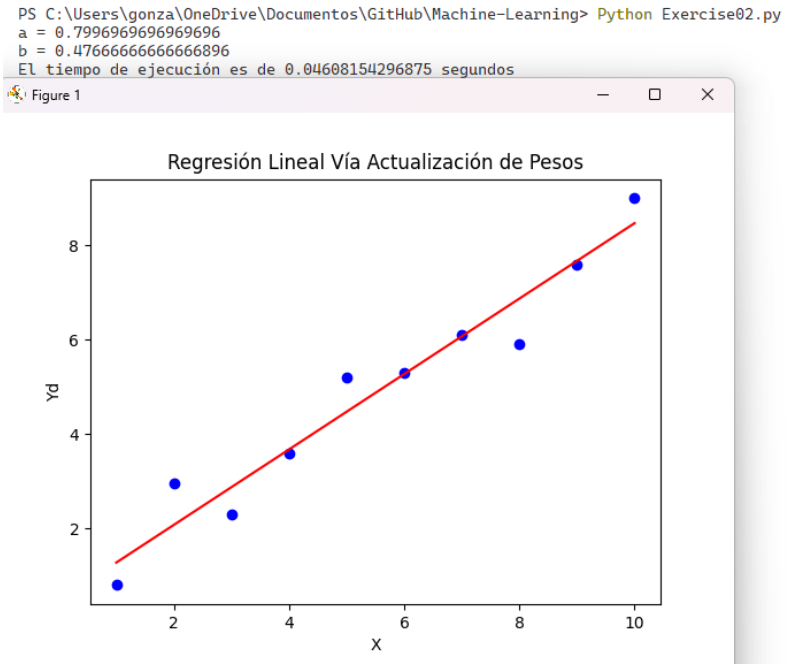


Figura 1.1. Pruebas con el primer learning rate. Fuente: Elaboración propia.

Para un lerning rate de **0.02** tenemos como resultado la Figura 1.2

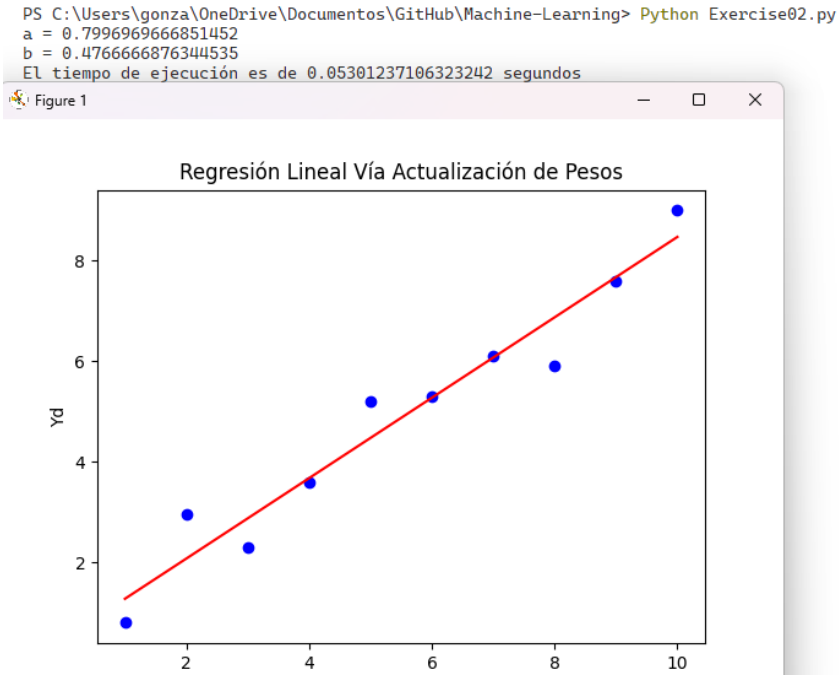


Figura 1.2. Pruebas con el segundo learning rate. Fuente: Elaboración propia.

Para un lerning rate de **0.001** tenemos como resultado la Figura 1.3

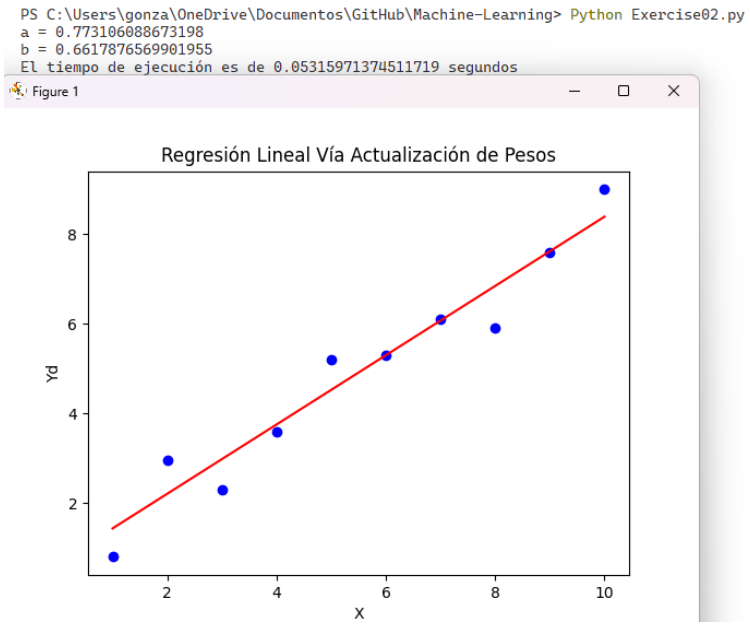


Figura 1.3. Pruebas con el tercer learning rate. Fuente: Elaboración propia.

Para el caso de las formulas obtenemos como resultado lo que se muestra en la Figura 1.4. Fue necesario realizar las pruebas en varias ocasiones para obtener un tiempo de ejecución adecuado, ya que este tipo de pruebas dependen mucho de la maquina en donde se ejecuten los códigos.

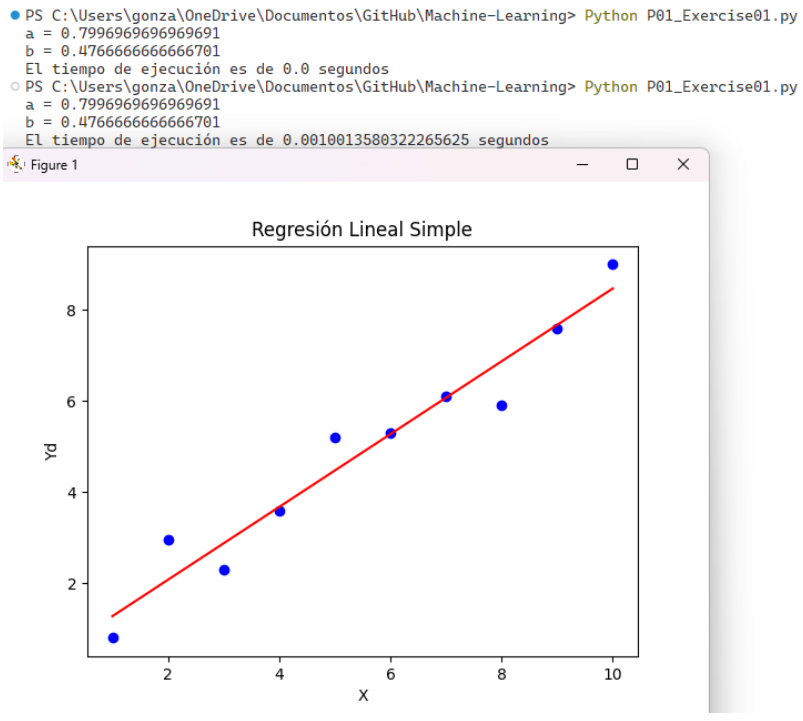


Figura 1.4. Pruebas con las fórmulas de regresión lineal simple. Fuente: Elaboración propia.

1.2.2. Ejercicio B

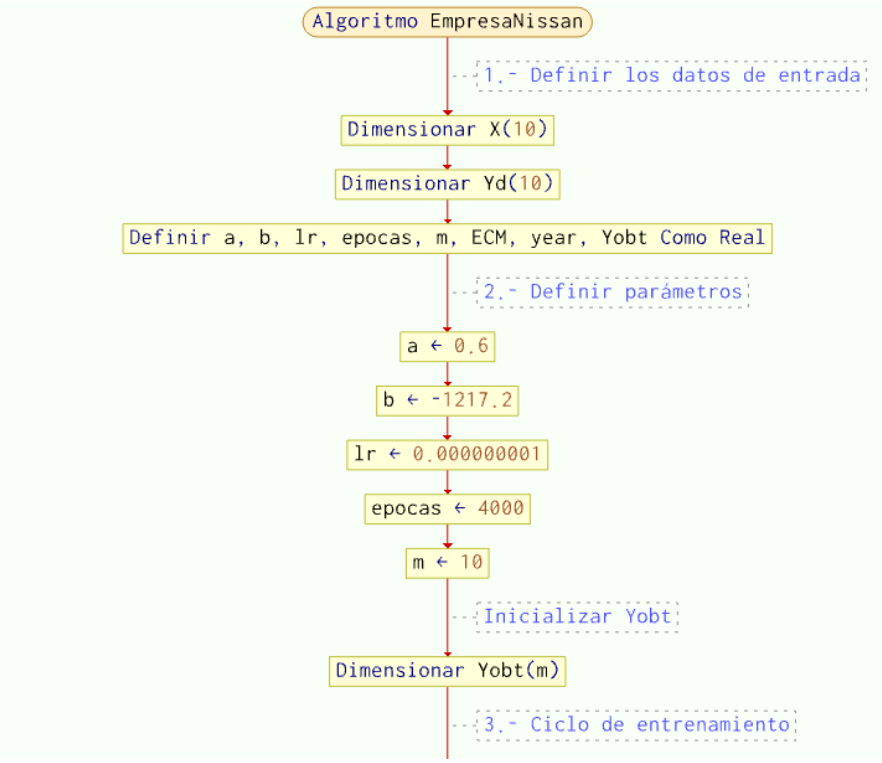
Para este segundo ejercicio se tiene una comercializadora de coches llamada Nissan que ha obtenido las siguientes ventas por año:

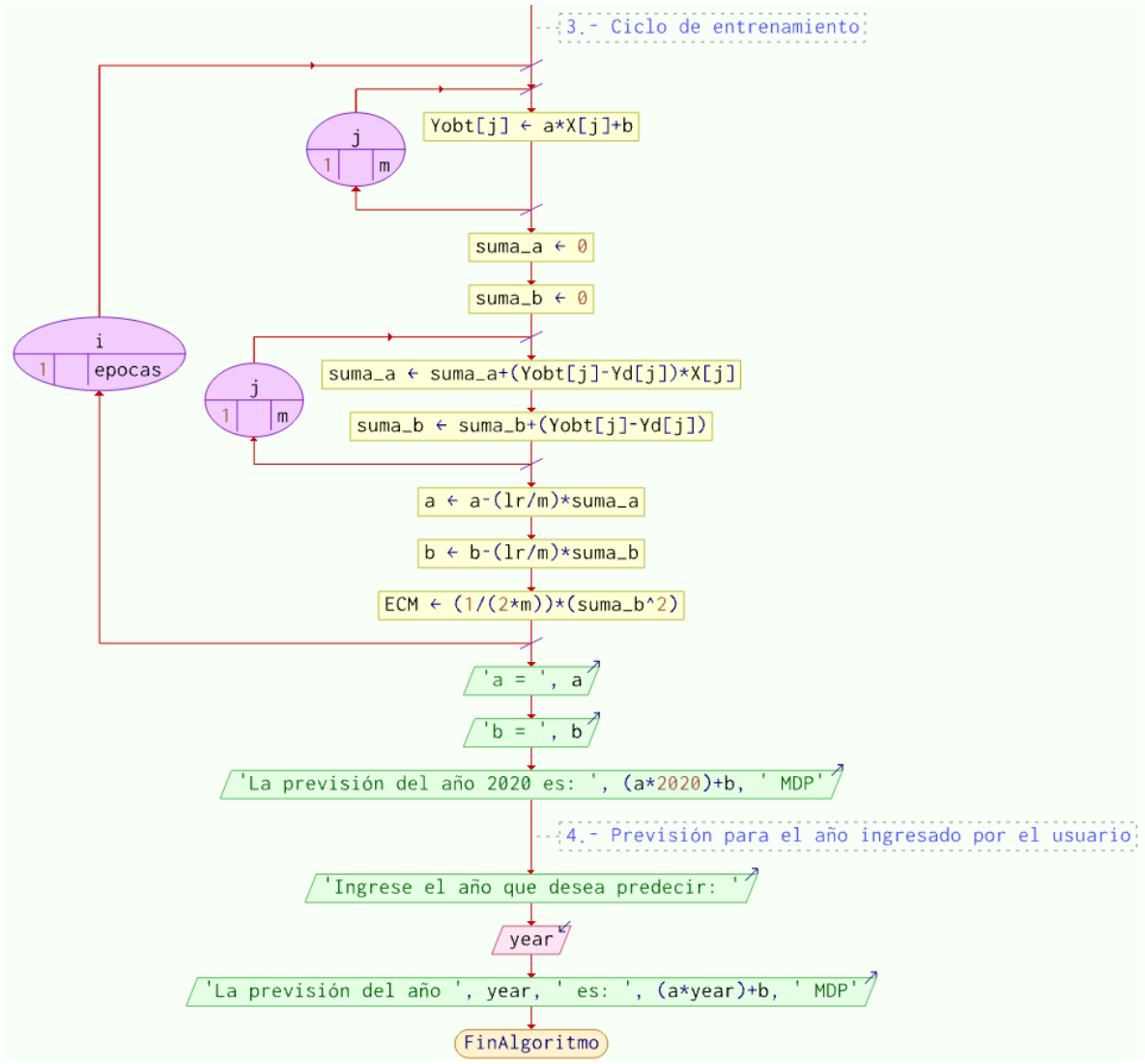
X (Años)	Yd (MDD)
2010	5.1
2011	5.4
2012	5.7
2013	6.0
2014	6.9
2015	7.5
2016	8.8
2017	9.8
2018	10.5
2019	9.0

Y se busca dar solución a la problemática de poder predecir el ingreso anual en MDD del 2020 o cualquier año posterior.

a) Diagrama de flujo

Para este ejercicio utilizamos la regresión lineal mediante el descenso de gradiente, para esto fue necesario implementar el diagrama de flujo que se muestra en figuras posteriores.




3.- Ciclo de entrenamiento
j
1
m
Yobt[j] ← a*X[j]+b
suma_a ← 0
suma_b ← 0
i
1
epocas
j
1
m
suma_a ← suma_a+(Yobt[j]-Yd[j])*X[j]
suma_b ← suma_b+(Yobt[j]-Yd[j])
a ← a-(lr/m)*suma_a
b ← b-(lr/m)*suma_b
ECM ← (1/(2*m))*(suma_b^2)
'a = ', a
'b = ', b
'La previsión del año 2020 es: ', (a*2020)+b, ' MDP'
4.- Previsión para el año ingresado por el usuario
'Ingrese el año que desea predecir: '
year
'La previsión del año ', year, ' es: ', (a*year)+b, ' MDP'
FinAlgoritmo

b) Código

```
import numpy as np
import matplotlib.pyplot as plt

# 1.- Definir los datos de entrada
X = np.array([2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018,
2019])
Yd = np.array([5.1,5.4, 5.7, 6, 6.9, 7.5, 8.8, 9.8, 10.5, 9.0])

# 2.- Definir parametros
a = 0.6
b = -1217.2
lr = 0.0000001
epocas = 4000
m = len(Yd)

Yobt = np.zeros(m)
for i in range(epocas):
   # 3.- Calculamos Yobt
   Yobt = a * X + b
   a -= (lr / m) * np.sum((Yobt - Yd) * X)
   b -= (lr / m) * np.sum((Yobt - Yd))
   ECM = (1 / (2 * m)) * np.sum(Yobt - Yd) **2

print(f"a = { a }")
print(f"b = { b }")

print(f"La prevision del año 2020 es: {(a*2020)+b} MDP")

#4.- Valores del usuario
year = float(input("Ingrese el año que desea predecir: "))
print(f"La prevision del año {year} es: {(a*year)+b} MDP")

#5.- Grafica de las funciones
plt.scatter(X, Yd, color='blue', label='Datos originales')
plt.plot(X, Yobt, color='red', label='Recta de regresión')

plt.xlabel('Año')
plt.ylabel('MDD')
plt.title('Regresión Lineal Vía Gradiente Descendente')
plt.show()
```

c) Funcionamiento

La Figura 1.5 muestra el resultado obtenido para los valores de a y b, así como la predicción para el año 2020 en MDD. También se le pregunta al usuario cual es el año que desea predecir.

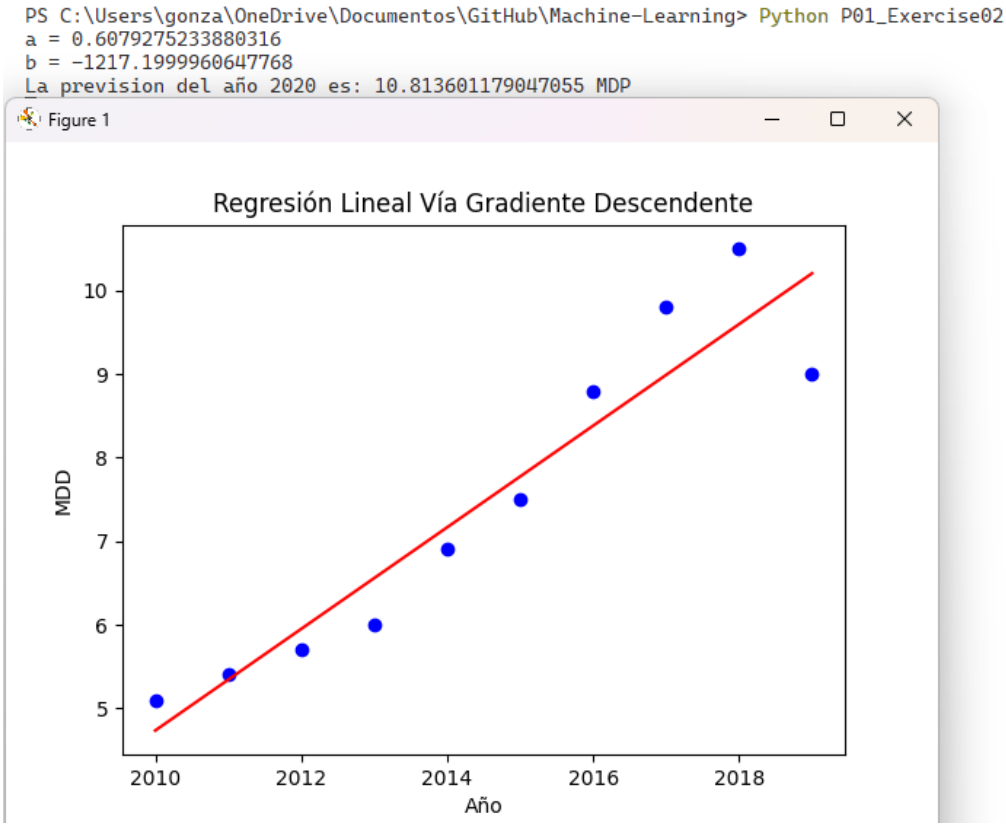


Figura 1.5. Pruebas para la predicción del año 2020. Fuente: Elaboración propia.

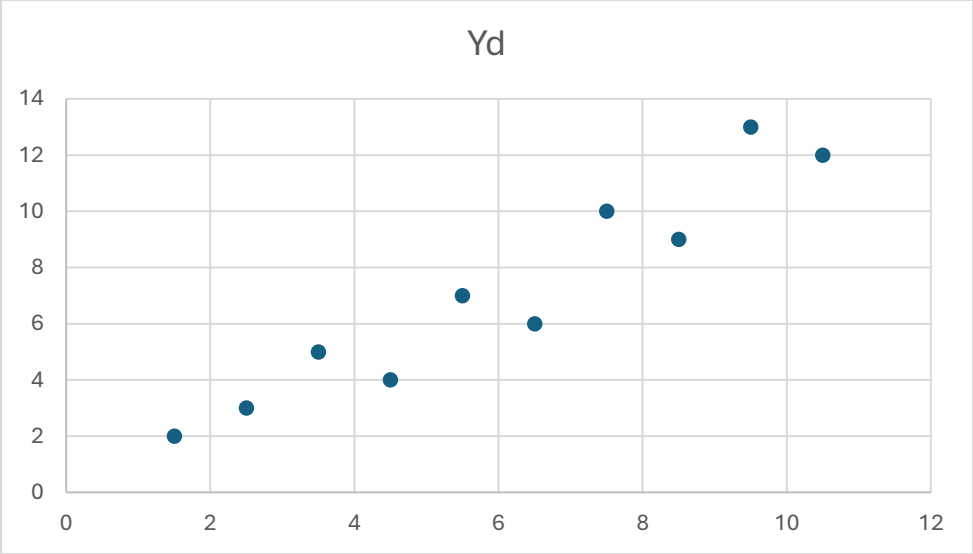
Posteriormente, en la Figura 1.6 le proporcionamos un valor del 2030 para obtener una predicción a largo plazo.

Figura 1.6. Pruebas para la predicción del año 2030. Fuente: Elaboración propia.

1.2.3. Ejercicio C

Sea:

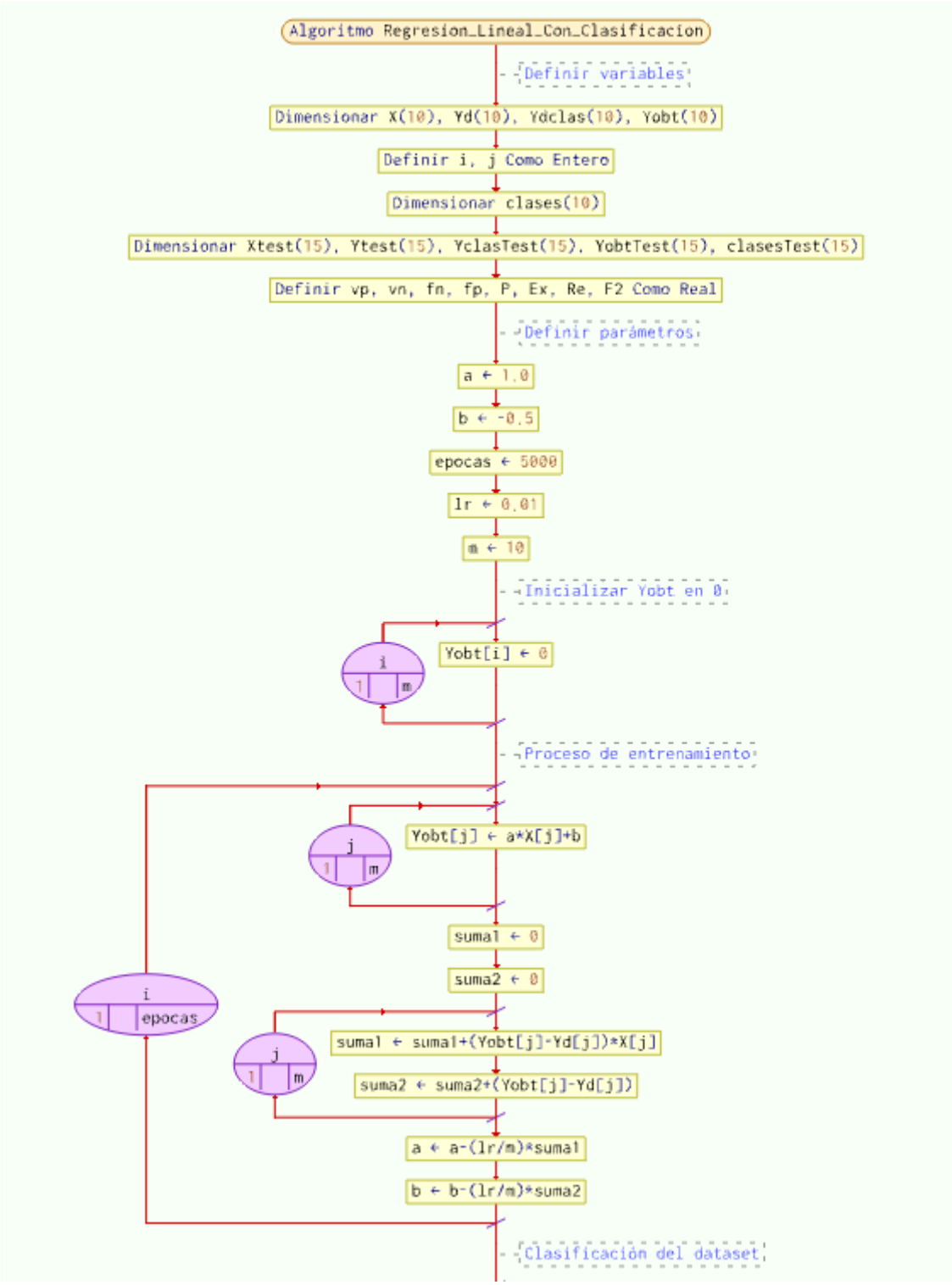
X	Yd
1.5	2
2.5	3
3.5	5
4.5	4
5.5	7
6.5	6
7.5	10
8.5	9
9.5	13
10.5	12

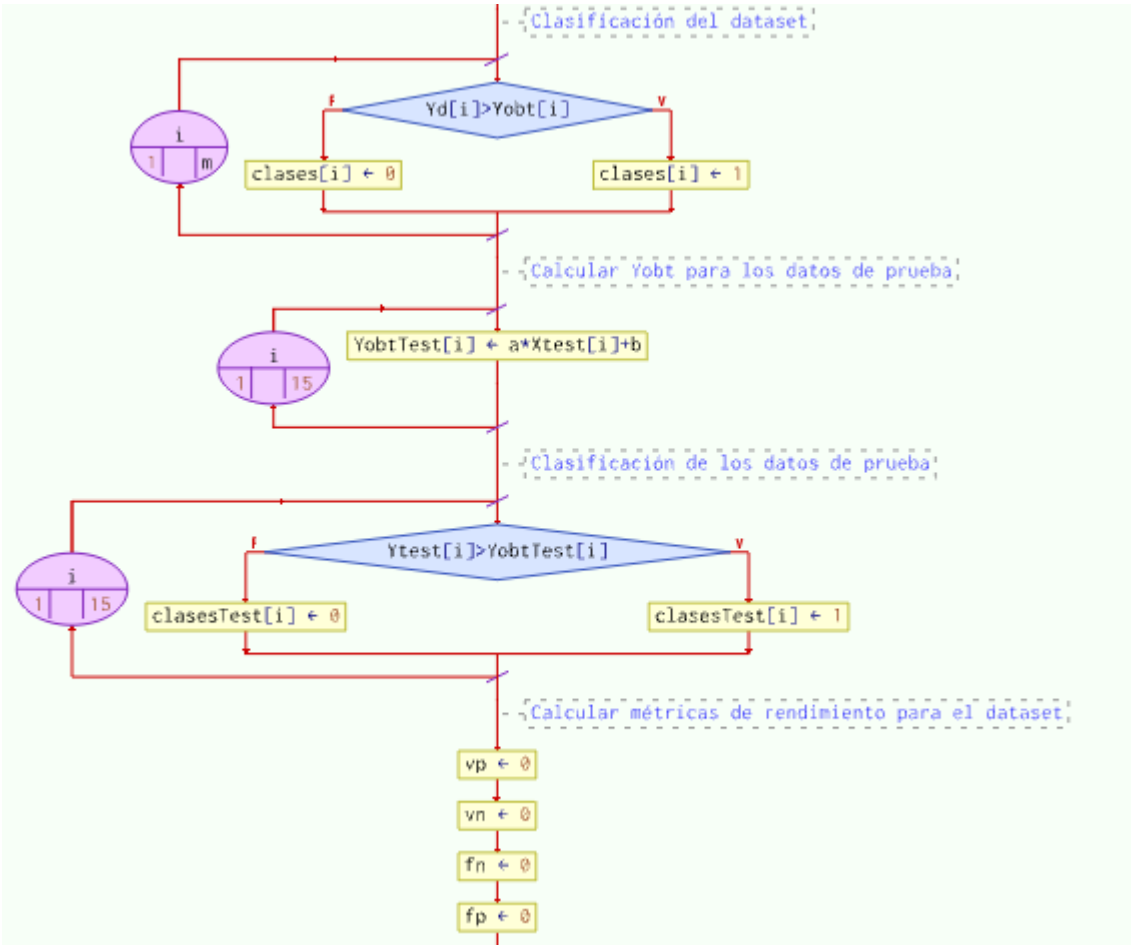


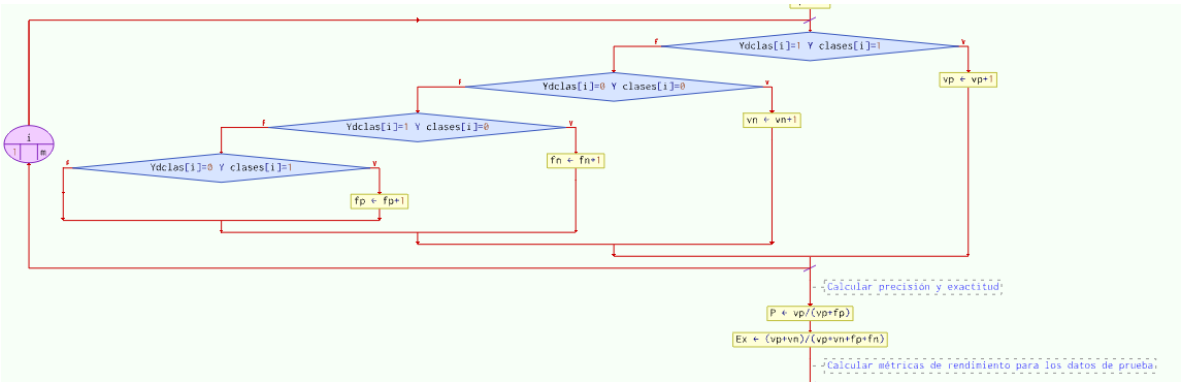
1. Obtener la curva $Yobt$ que minimice el ECM
2. Dividir el plano en 2 clases (0 y 1). El programa debe clasificar cada valor de $Yobt$ en alguna de las dos clases.
3. Utilizando únicamente **Datatest**. Asigne 15 nuevos puntos $(X, Yd)_{test}$ y clasifíquelos como en punto anterior.
4. Determinar precisión, exactitud y F2 Score.

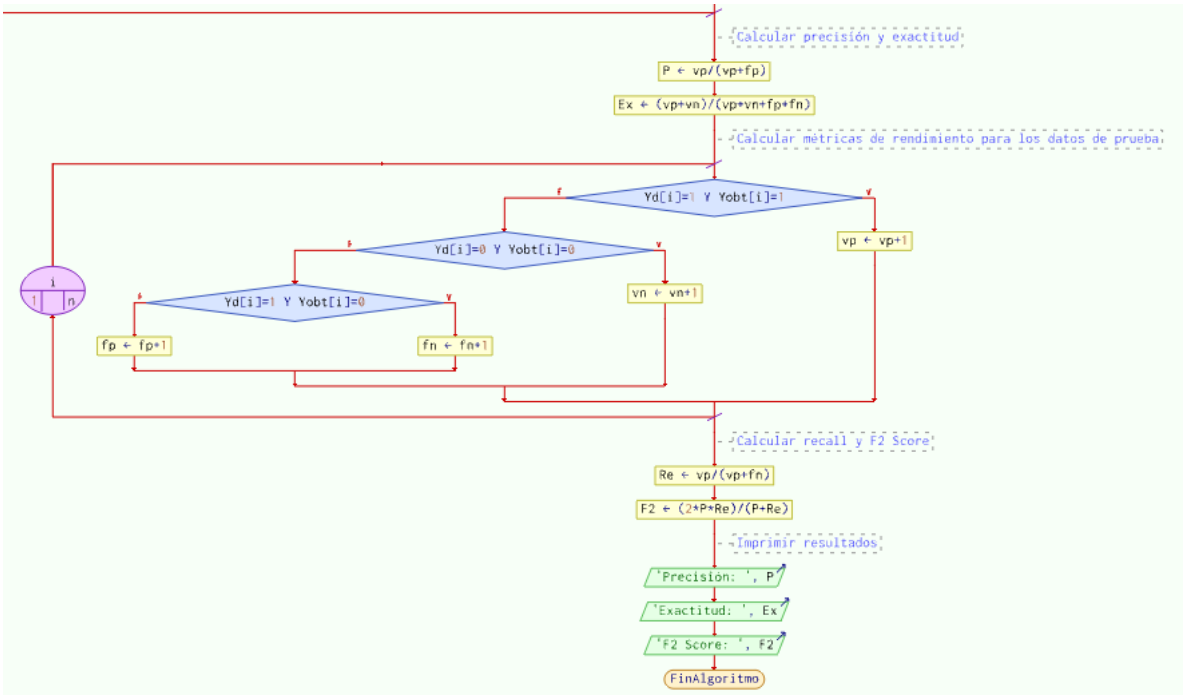
Para implementar este ejercicio utilizamos la regresión lineal mediante el descenso de gradiente y obtuvimos cada uno de los puntos como se muestra en el diagrama de flujo y el código implementado.

a) Diagrama de flujo




Clasificación del dataset
i
1
m
Yd[i]>Yobt[i]
F
V
clases[i] ← 0
clases[i] ← 1
Calcular Yobt para los datos de prueba
i
1
15
YobtTest[i] ← a*Xtest[i]+b
Clasificación de los datos de prueba
i
1
15
Ytest[i]>YobtTest[i]
F
V
clasesTest[i] ← 0
clasesTest[i] ← 1
Calcular métricas de rendimiento para el dataset
vp ← 0
vn ← 0
fn ← 0
fp ← 0


Ydclas[i]=1 Y clases[i]=1
vp ← vp+1
Ydclas[i]=0 Y clases[i]=0
vn ← vn+1
Ydclas[i]=1 Y clases[i]=0
fn ← fn+1
Ydclas[i]=0 Y clases[i]=1
fp ← fp+1
Calcular precisión y exactitud
P ← vp/(vp+fp)
Ex ← (vp+vn)/(vp+vn+fp+fn)
Calcular métricas de rendimiento para los datos de prueba



b) Código

```
import numpy as np
import matplotlib.pyplot as plt

def metricas_rend(Yobt, Yd):
  metricas = []
  vp = 0
  vn = 0
  fn = 0
  fp = 0

  tamanioYd = len(Yd)
  tamanioYobt = len(Yobt)

  if tamanioYd == tamanioYobt:
      for i in range(tamanioYd):
            if Yd[i] == 1 and Yobt[i] == 1:
            vp = vp + 1
            elif Yd[i] == 0 and Yobt[i] == 0:
            vn = vn + 1
            elif Yd[i] == 1 and Yobt[i] == 0:
            fn = fn + 1
            elif Yd[i] == 0 and Yobt[i] == 1:
            fp = fp + 1

  metricas = [vp, vn, fn, fp]

  return metricas
```

```
def asignar_clase(Yobt, Yd):
      clases = np.zeros(len(Yd))
      for i in range(len(Yobt)):
      if Yd[i] > Yobt[i]:
            clases[i] = 1
      else:
            clases[i] = 0

      return clases

# 1.- Definir los datos de entrada
X = np.array([1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5, 10.5])
Yd = np.array([2, 3, 5, 4, 7, 6, 10, 9, 13, 12])
Ydclas = np.array([1, 1, 0, 0, 1, 1, 0, 1, 0, 1])

# 2.- Definir parametros
a = 1.0
b = -0.5
epocas = 5000
lr = 0.01
m = len(Yd)
Yobt = np.zeros(m)


for i in range(epocas):
      # 3.- Calculamos Yobt
      Yobt = a * X + b
      a -= (lr / m) * np.sum((Yobt - Yd) * X)
      b -= (lr / m) * np.sum(Yobt - Yd)
      ECM = (1 / (2 * m)) * np.sum((Yobt - Yd)**2)

print(f"Yobt = {Yobt}\n")

# 4.- Clasificacion de los dataset
clases = asignar_clase(Yobt, Yd)
print(f"Clasificacion Yobt: { clases }")

# 5.- Asignamos 15 datatest
Xtest = np.array([11.5, 12.5, 13.5, 14.5, 15.5, 16.5, 17.5, 18.5, 19.5,
20.5, 21.5, 22.5, 23.5, 24.5, 25.5])
Ytest = np.array([14, 11, 13, 16, 19, 20, 24, 21, 26, 22, 28, 30, 32, 34,
36])
YclasTest = np.array([1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1])

YobtTest = a * Xtest + b

# 6.- Clasificacion de los datatest
clasesTest = asignar_clase(YobtTest, Ytest)
print(f"Clasificacion Datatest: { clasesTest }")
```

```
# 7.- Determinar Presicion y Exactitud
metricasYds = metricas_rend(clases, Ydclas)

# metricas = [vp,vn,fn,fp]
P = metricasYds[0] / (metricasYds[0] + metricasYds[3])
print(f"Precision = {P}")
Ex = (metricasYds[0] + metricasYds[1]) / (metricasYds[0] + metricasYds[1]
+ metricasYds[3] + metricasYds[2])
print(f"Exactitud = {Ex}")

# Determinar Recall
metricasYdt = metricas_rend(clasesTest, YclasTest)
metricasT = np.add(metricasYds, metricasYdt)
Re = metricasT[0] / (metricasT[0] + metricasT[2])

# 8.- Determinar F2 Score
F2 = (2 * P * Re) / (P + Re)
print(f"F2 Score: { F2 }")

# 9.- Graficar funciones
fig, (ax1, ax2) = plt.subplots(2, 1)

# Agregar títulos a cada subplot
ax1.set_title("Gráfica Dataset")
ax1.scatter(X, Yd)
ax1.plot(X, Yobt, color="red")

ax2.set_title("Gráfica Datatest")
ax2.scatter(Xtest, Ytest)
ax2.plot(Xtest, YobtTest, color="red")

# Mostrar las gráficas
plt.tight_layout()  # Asegura que los títulos no se sobrepongan
plt.show()
```

c) Funcionamiento

La Figura 1.7 muestra el resultado obtenido para la precisión, exactitud y recall.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\gonza\OneDrive\Documentos\GitHub\Machine-Learning\Practice01> Python P01_Exercise03.py
Yobt = [ 1.72723903  2.92118419  4.11512935  5.30907451  6.50301967  7.69696483
  8.89090999 10.08485515 11.27880031 12.47274547]

Clasificacion Yobt: [1. 1. 1. 0. 1. 0. 1. 0. 1. 0.]
Clasificacion Datatest: [1. 0. 0. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1.]
Precision = 0.5
Exactitud = 0.4
F2 Score: 0.5
```

Figura 1.7. Cálculo de precisión, Exactitud y F2 Score. Fuente: Elaboración propia.

Posteriormente, se muestra en la figura 1.8 se muestran la gráfica para los valores dataset para la visualización de la línea de regresión lineal con respecto a los valores esperados.

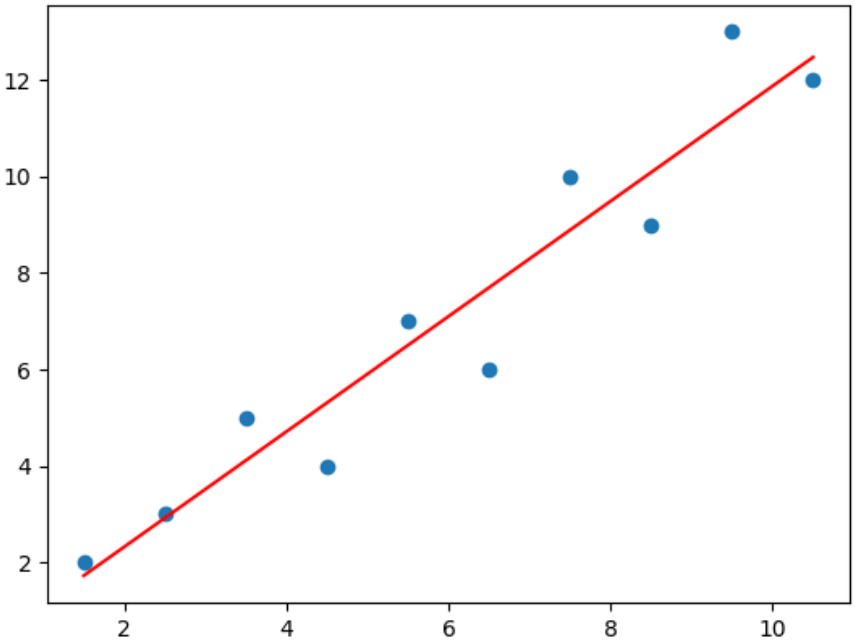


Figura 1.8. Gráfica de ajuste del modelo de regresión lineal dataset. Fuente: Elaboración propia.

A continuación, en la imagen 1.9. se mostrará la gráfica para los valores de datatest para la visualización de la línea de regresión lineal con respecto a los valores esperados de acuerdo el modelo

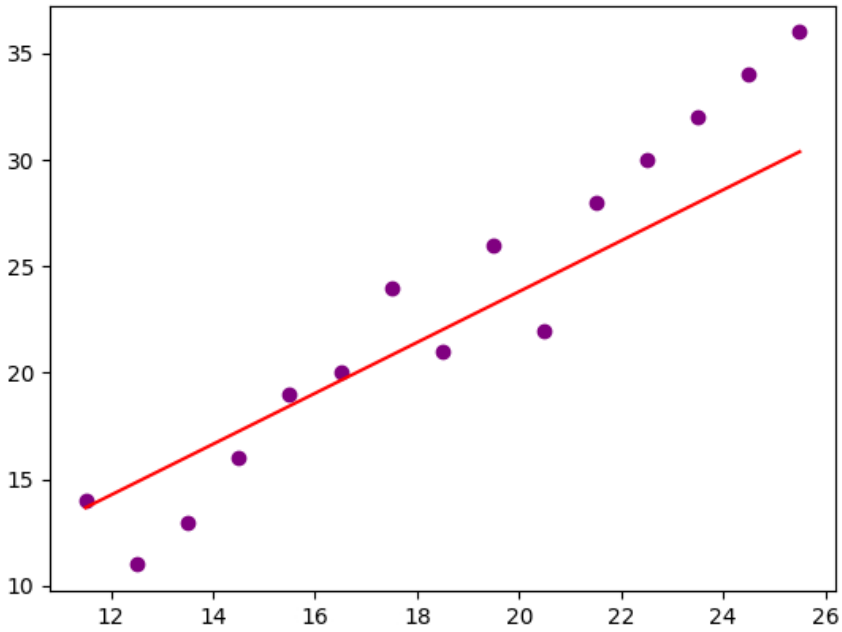


Figura 1.9. Gráfica de ajuste del modelo de regresión lineal datatest. Fuente: Elaboración propia.

En la figura 1.10. se muestran ambas graficas de dataset y datatest para la comparación del comportamiento del modelo de acuerdo con los dos grupos de datos.

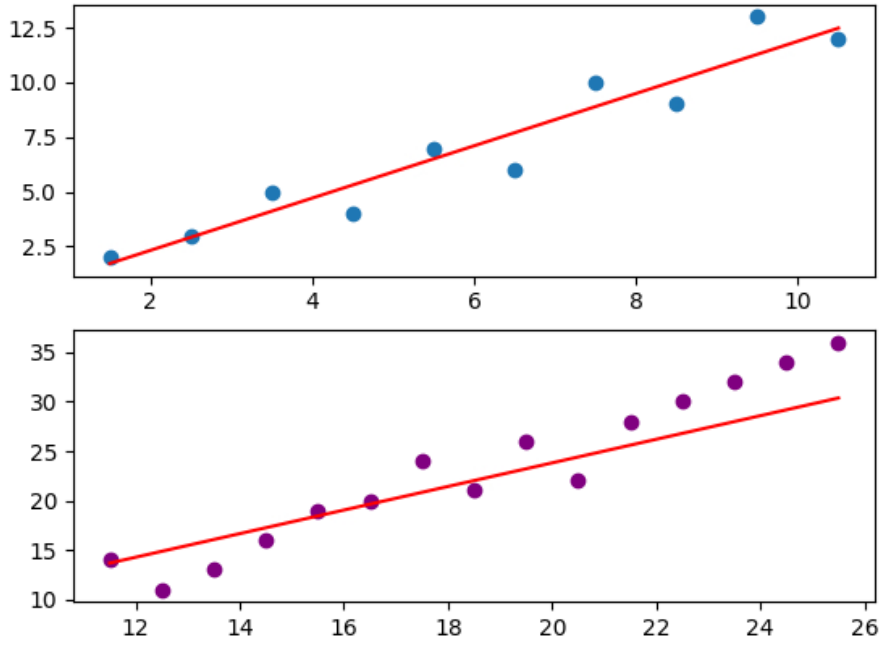


Figura 1.10. Gráfica de ajuste del modelo de regresión lineal datatest. Fuente: Elaboración propia.

1.3. CONCLUSIÓN

En la practica 1 se trabajaron modelos de regresión lineal simple y por actualización de pesos donde en el ejercicio **A** se realiza una comparación entre estos 2 modelos respecto al tiempo de proceso que tardan en ejecutar estos modelos para la obtención de los valores de a y b, donde en la imagen 1.11 se pueden ver los resultados de modelo de regresión lineal por actualización de pesos y en la imagen 1.12 los de regresión lineal simpe

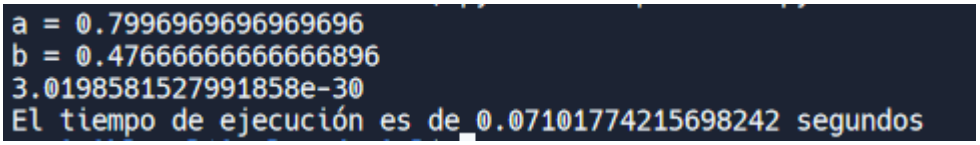


Figura 1.11. Valores de a y b con tiempo de ejecución, en actualización de pesos. Fuente: Elaboración propia.

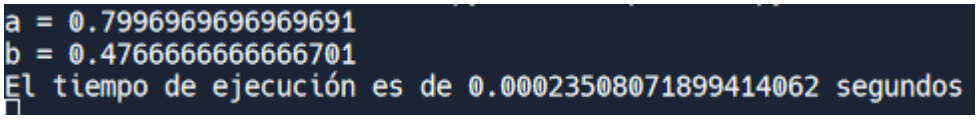


Figura 1.12. Valores de a y b con tiempo de ejecución, regresión lineal simple. Fuente: Elaboración propia.

Como se puede apreciar en las imágenes, los tiempos de ejecución entre la actualización y de pesos y regresión lineal siempre se puede apreciar que la regresión lineal siempre es más rápida de realizar dado que tarda 0.00023s contra 0.07s en la actualización de pesos, esto se debe a que en la actualización de pesos se está iterando n cantidad de veces, para este caso fueron 4000 veces las que si itero para obtener los valores de a y b, en cambio en la regresión lineal simple se utiliza una formula lo cual evita la iteración por lo tanto reduce el tiempo de ejecución , y ambos tuvieron los valores de a y b muy parecidos dado que solo varían entre el digito 16 y 15 por lo cual se puede decir que ambos cumplen con la obtención de estos valores. Los modelos fueron ejecutados de 2 maneas distintas

La primera manera fue en una laptop con las siguientes características, procesador AMD Ryzen 5 4600H with Radeon Graphics a 3.0 Ghz con 16GB de RAM a una velocidad de 3200 MT/s en formato de 2 modulos de 8GB cada uno con un sistema operativo Windows 11 en la versión Home 23H2. Donde los tiempos de ejecución daban 0.0s la mayoría de las veces.

La segunda manera fue por medio de Replit, una plataforma de desarrollo en línea la cual permite ejecutar el código directamente desde el navegador. Los recursos CPU y RAM los proporciona como recursos compartidos en la nube, el código es ejecutado en un entorno virtual debido a esto los tiempos de ejecución dependen de la carga del servidor y de las limitaciones del entorno compartido, los tiempos de ejecución de los modelos en Replit son los mostrados en las figuras 1.11 y 1.12 debido a que siempre mostraba tiempos de ejecución en comparación con la ejecución en el dispositivo portátil.

INSTITUTO POLITÉCTICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

UNIDAD DE APRENDIZAJE:

MACHINE LEARNING

PRÁCTICA 2: Regresión Logística

INTEGRANTES:

Hernández Hernández Roberto Isaac

Gonzalez Llamosas Noe Ramses

PROFESOR:

Ortiz Castillo Marco Antonio

FECHA DE ENTREGA: 14/10/2023

2. PRACTICA: REGRESIÓN LOGÍSTICA

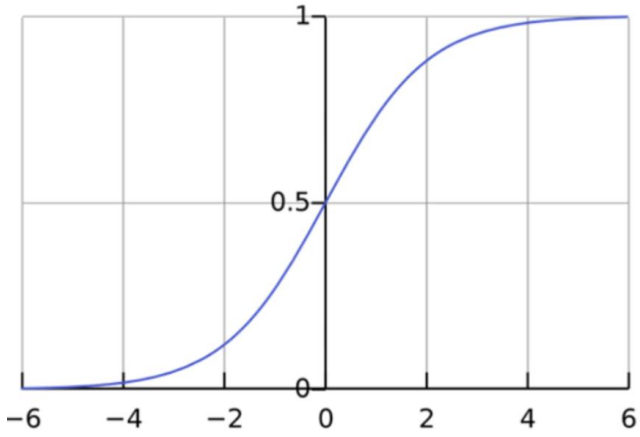
2.1. INTRODUCCIÓN

La regresión logística es una técnica de análisis de datos que utiliza las matemáticas para encontrar las relaciones entre dos factores de datos. Luego, utiliza esta relación para predecir el valor de uno de esos factores basándose en el otro. Normalmente, la predicción tiene un número finito de resultados, como un sí o un no. La regresión logística es una técnica importante en el campo de la inteligencia artificial y el machine learning (AI/ML). Los modelos de ML son programas de software que puede entrenar para realizar tareas complejas de procesamiento de datos sin intervención humana. Los modelos de ML creados mediante regresión logística ayudan a las organizaciones a obtener información procesable a partir de sus datos empresariales. Pueden usar esta información para el análisis predictivo a fin de reducir los costos operativos, aumentar la eficiencia y escalar más rápido [3].

La regresión logística es una de las diferentes técnicas de análisis de regresión que los científicos de datos utilizan habitualmente en machine learning (ML). Para entender la regresión logística, primero debemos entender el análisis de regresión básica. A continuación, utilizamos un ejemplo de análisis de regresión lineal para demostrar cómo funciona el análisis de regresión. La regresión logística es un modelo estadístico que utiliza la función logística, o función logit, en matemáticas como la ecuación entre *x* e*y*. La función logit mapea *y* como una función sigmoidea de *x* [3].

$$f(x) = \frac{1}{1 * e^{-x}}$$

Si representa esta ecuación de regresión logística, obtendrá una curva en S como la que se muestra a continuación.



2.2. DESARROLLO

2.2.1. Ejercicio A

a) Demostración

Suponga que propone un Z como $Z = X_1\theta_1 + X_2\theta_2 + X_3\theta_3 + \cdots + X_n\theta_n + \theta_0 = \theta^T X + \theta_0$ donde θ_0 se le llama Bias y es una variable que permite, ajustar los hiperplanos en cualquier posición, determine a través de descenso de gradiente la forma en la que debe actualizarse θ_0 para una regresión logística.

Considerando la ecuación 1 donde:

$$\theta_0 = \theta_0 - \eta \frac{\partial J}{\partial \theta_0}$$

Es necesario determinar $\frac{\partial J}{\partial \theta_0}$ considerando que

$$J = -\frac{1}{m}\sum_{i=1}^{m}[\,Y_d * \ln(h) + (1 - Y_d) * \ln(1-h)] \;\&\&\; h = h - (1-h)$$

$$\frac{\partial J_1}{\partial \theta_0} = \frac{\partial [Y_{di} * \ln(h)]}{\partial \theta_0} = Y_{di}\left[\frac{\partial \ln(h)}{\partial \theta_0}\right] = Y_{di}\frac{\partial [\ln(h)]}{\partial \theta_0} = Y_{di}\frac{1}{h} * h(1-h)$$

$$\frac{\partial J_1}{\partial \theta_0} = Y_{di}(1-h)$$

$$\frac{\partial J_2}{\partial \theta_0} = \frac{\partial [(1 + Y_{di}) * \ln(1-h)]}{\partial \theta_0} = (1 + Y_{di}) * \frac{\ln(1-h)]}{\partial \theta_0}$$

$$\frac{\partial J_2}{\partial \theta_0} = (1 - Y_{di})\left(\frac{1}{1-h}\right) * \big(-h(1-h)\big) = (-h + hY_{di})$$

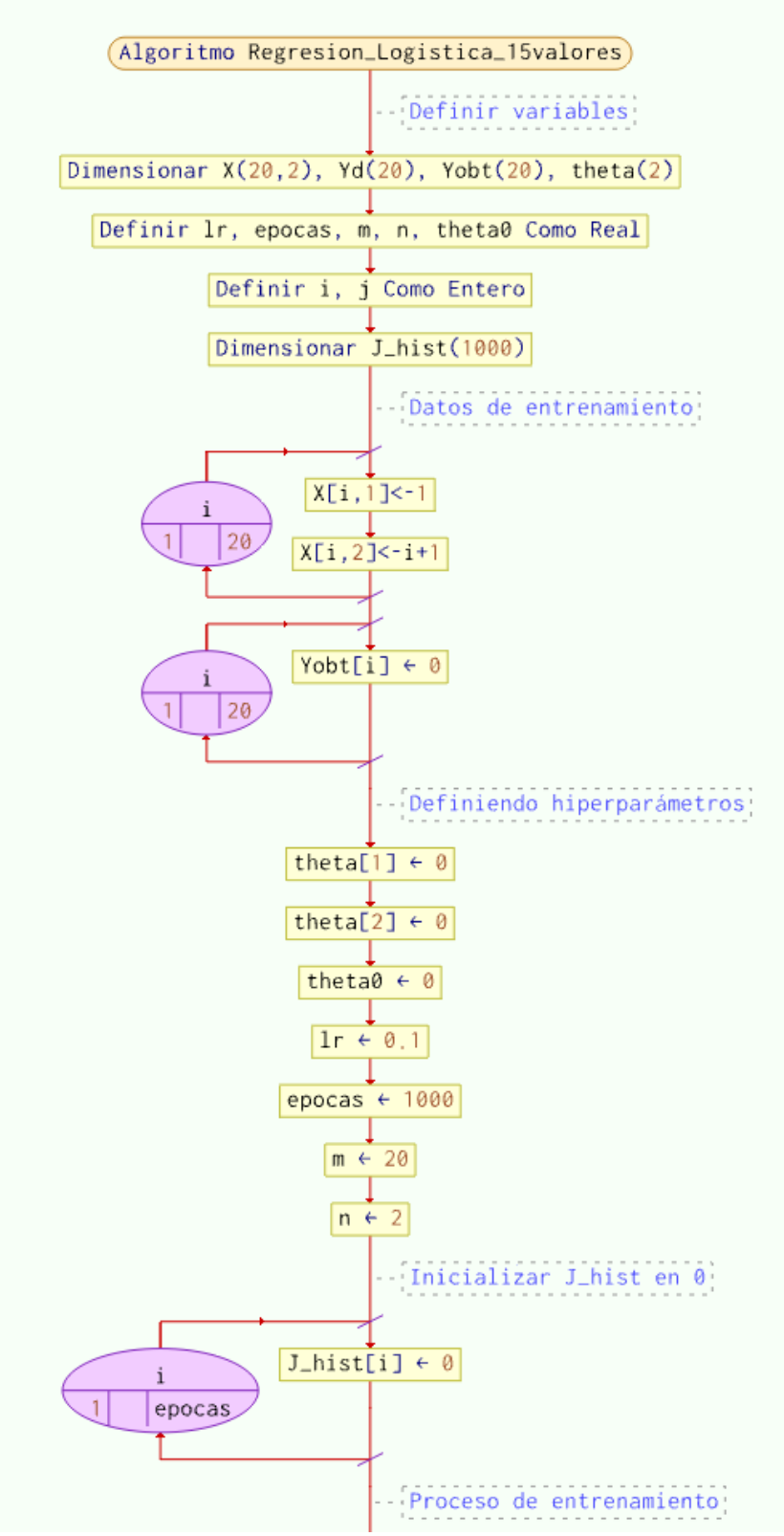
$$\frac{\partial J}{\partial \theta_0} = -\frac{1}{m}\sum_{i=1}^{m}[\,Y_{di}(1-h) - (-h + hY_{di})] - \frac{1}{m}\sum_{i=1}^{m}[\,Y_{di} - Y_{di}h + h + hY_{di}]$$

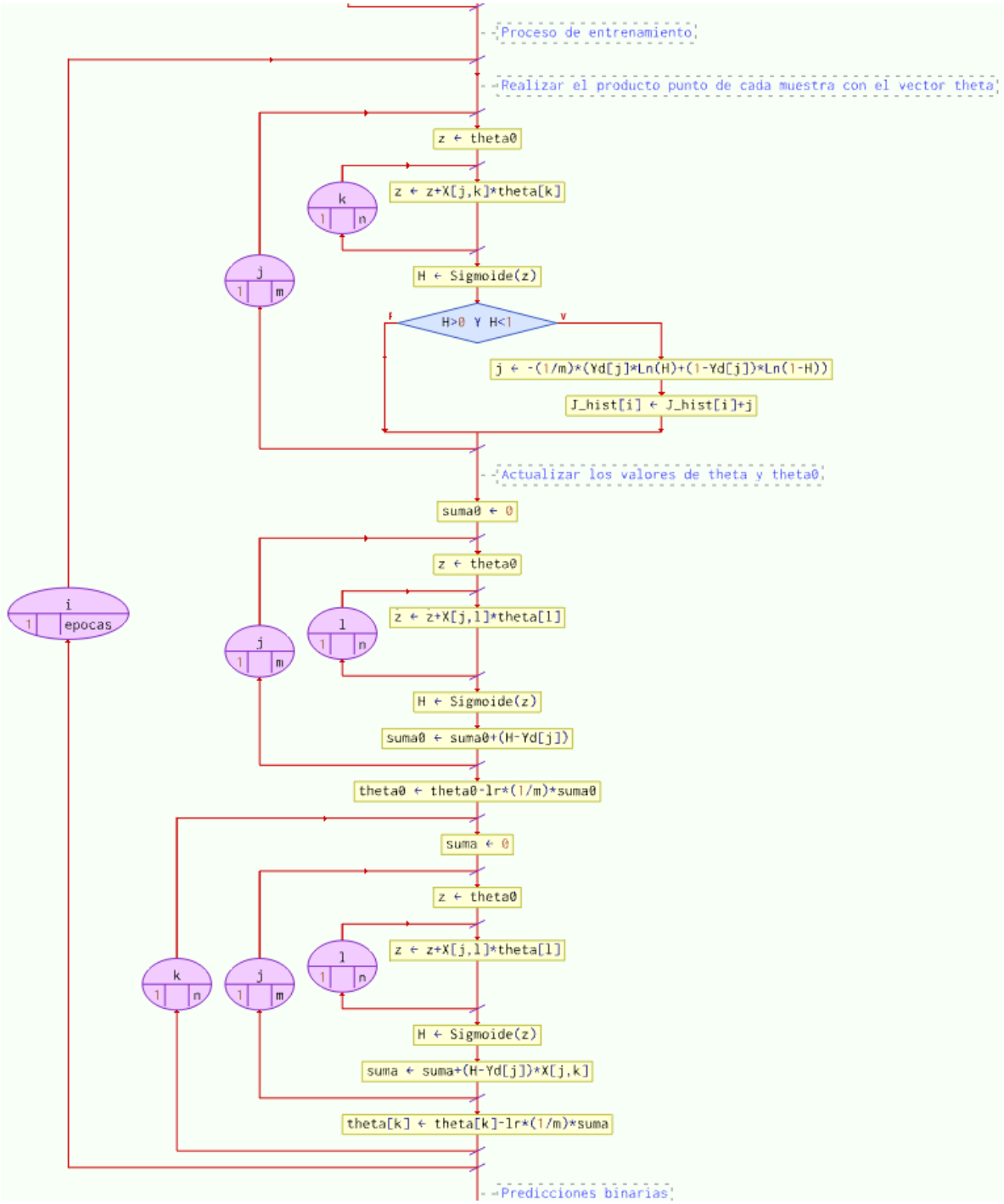
$$\frac{\partial J}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}[\,h - Y_{di}]$$

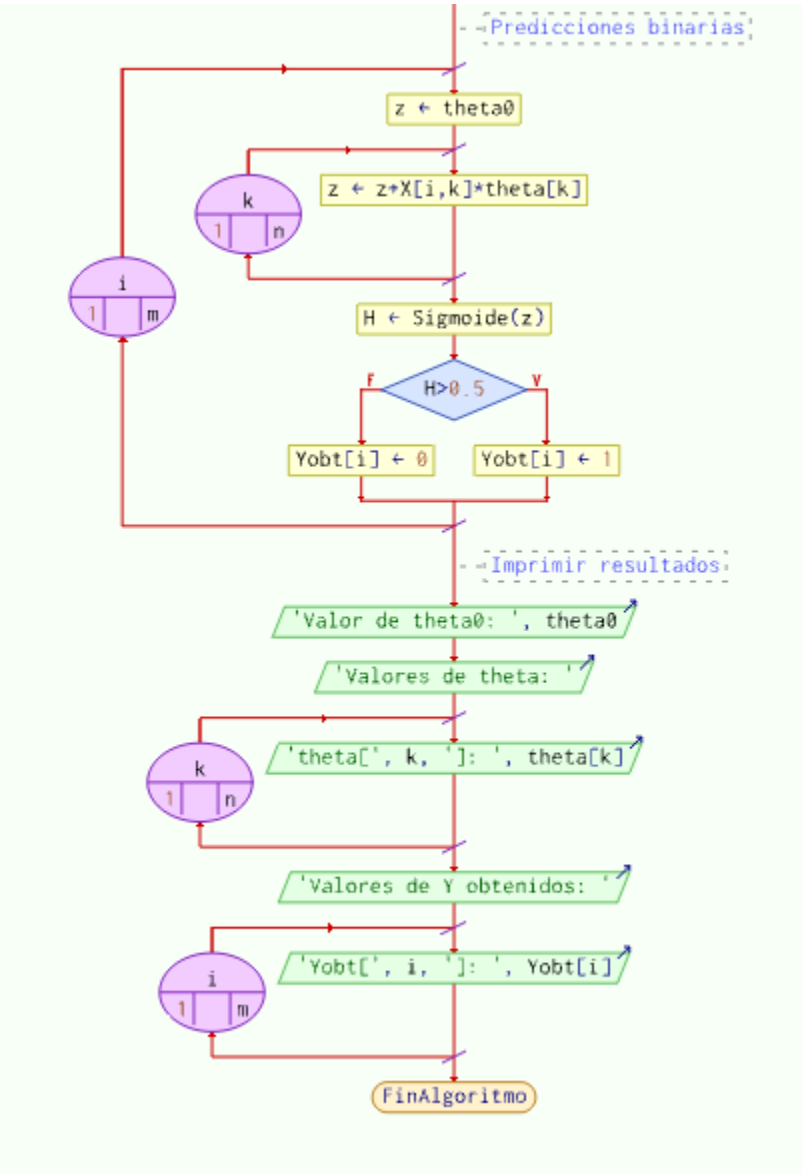
$$\theta_0 = \theta_0 - \eta\frac{1}{m}\sum_{i=1}^{m}[\,h - Y_{di}]$$

2.2.2. Ejercicio B

a) Diagrama de flujo




Proceso de entrenamiento
Realizar el producto punto de cada muestra con el vector theta
z ← theta0
k
1
n
z ← z+X[j,k]*theta[k]
j
1
m
H ← Sigmoide(z)
F
H>0 Y H<1
V
j ← -(1/m)*(Yd[j]*Ln(H)+(1-Yd[j])*Ln(1-H))
J_hist[i] ← J_hist[i]+j
Actualizar los valores de theta y theta0
suma0 ← 0
z ← theta0
i
1
epocas
l
1
n
z ← z+X[j,l]*theta[l]
j
1
m
H ← Sigmoide(z)
suma0 ← suma0+(H-Yd[j])
theta0 ← theta0-lr*(1/m)*suma0
suma ← 0
z ← theta0
l
1
n
z ← z+X[j,l]*theta[l]
k
1
n
j
1
m
H ← Sigmoide(z)
suma ← suma+(H-Yd[j])*X[j,k]
theta[k] ← theta[k]-lr*(1/m)*suma
Predicciones binarias



b) Código

```
import numpy as np
import matplotlib.pyplot as plt

# Definir la función sigmoide
def Sigmoide(z):
    return 1 / (1 + np.exp(-z))

# Corregir la función ValidateH para trabajar con arrays
def ValidateH(H):
    Predictions = np.zeros(len(H))
    for i in range(len(H)):
        if H[i] > 0.5:
```

```
            Predictions[i] = 1
        else:
            Predictions[i] = 0
    return Predictions

# Datos de entrenamiento
X =
np.array([[1,2],[1,3],[1,4],[1,5],[1,6],[1,7],[1,8],[1,9],[1,10],[1,11],[
1,12],[1,13],[1,14],[1,15],[1,16],[1,17],[1,18],[1,19],[1,20],[1,21]])
Yd = np.array([0,0,1,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,0])
Yobt = np.zeros(len(Yd))

# Definiendo hiperparámetros
n = len(X[0])
theta = np.zeros(n)
theta0 = 0

# Hiperparámetros dentro del modelo
lr = 0.1
epocas = 1000
m = len(Yd)

J_hist = np.zeros(epocas)

for i in range(epocas):
    Z = theta0 + np.dot(X, theta)
    H = Sigmoide(Z)

    # Evaluar la función de costo
    J = -(1/m) * np.sum(Yd * np.log(H) + (1 - Yd) * np.log(1 - H))

    J_hist[i] = J

    theta0 = theta0 - lr * (1/m) * np.sum(H - Yd)
    theta = theta - lr * (1/m) * np.dot(X.T, (H - Yd))

Yobt = ValidateH(H)

print(f"Theta0: {theta0}")
print(f"Theta: {theta}")

plt.scatter(X[:, 1], Yd, color='blue', label='Datos de entrenamiento')

J_hist_scaled = (J_hist - np.min(J_hist)) / (np.max(J_hist) -
```

```
np.min(J_hist))
plt.plot(np.linspace(2, 21, epocas), J_hist_scaled, color='green',
label='Función de costo J (escalada)')

plt.xlabel('X')
plt.ylabel('Yd / H')
plt.title('Regresión logística')
plt.legend()
plt.show()
```

c) Funcionamiento

La Figura 1.13 muestra el funcionamiento de la regresión logística mediante el código previamente mostrado. Es fundamental observar el comportamiento de la función de costo con respecto al learning rate elegido y los valores obtenidos en theta y theta0.

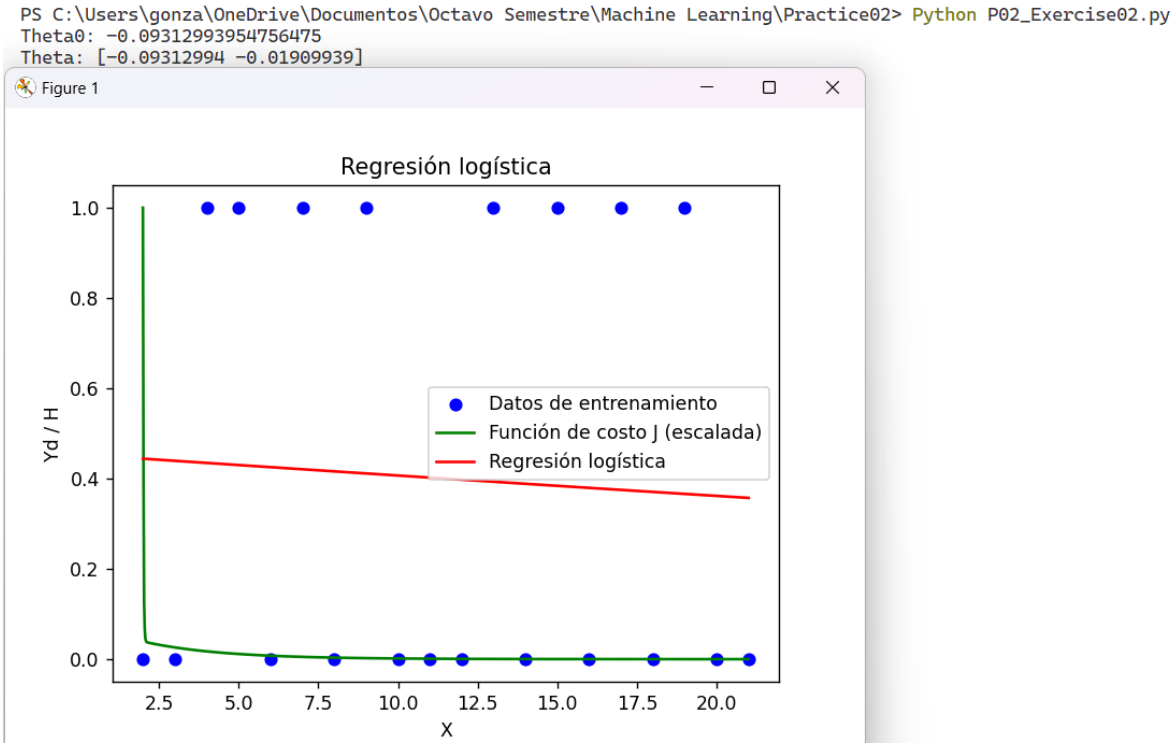
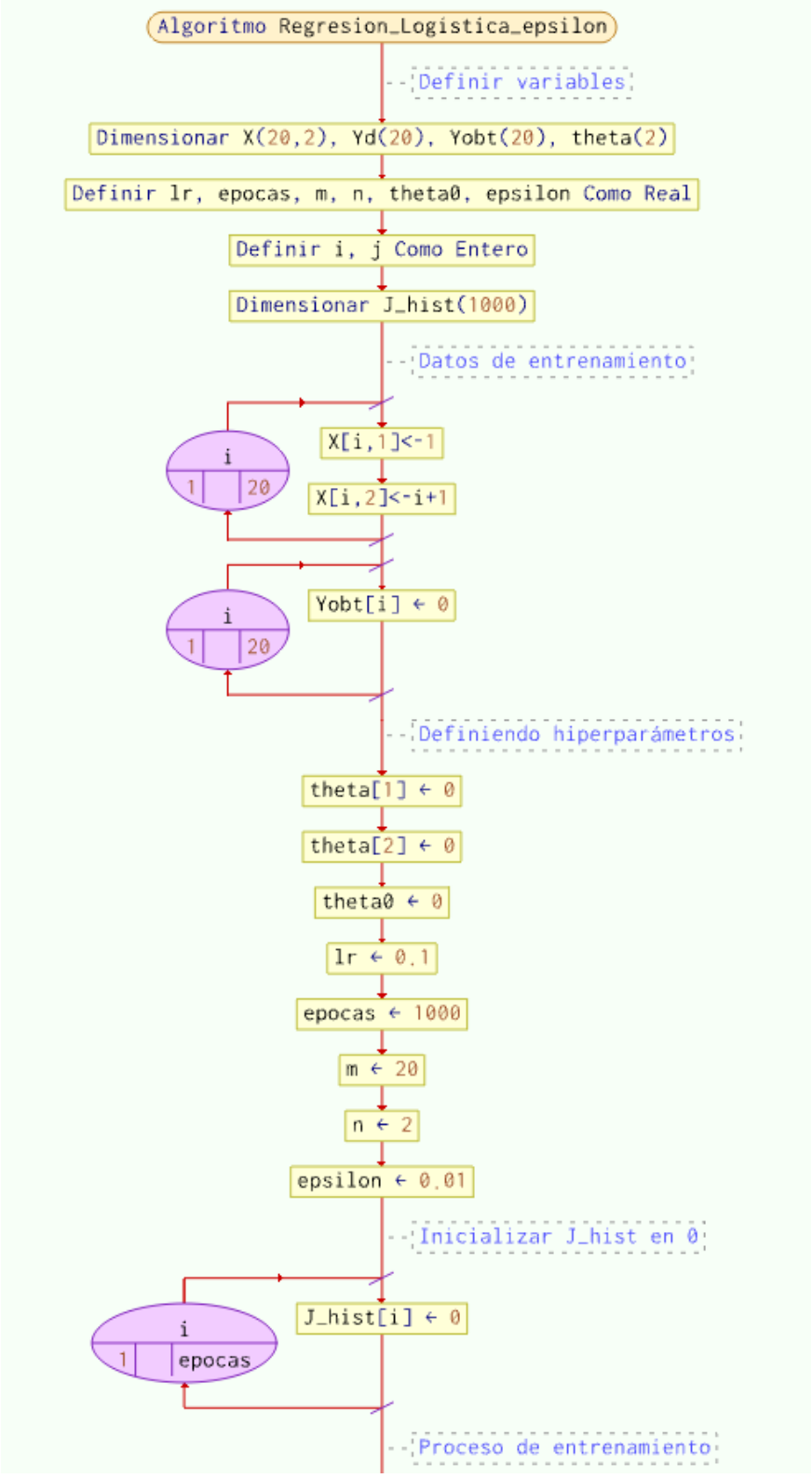
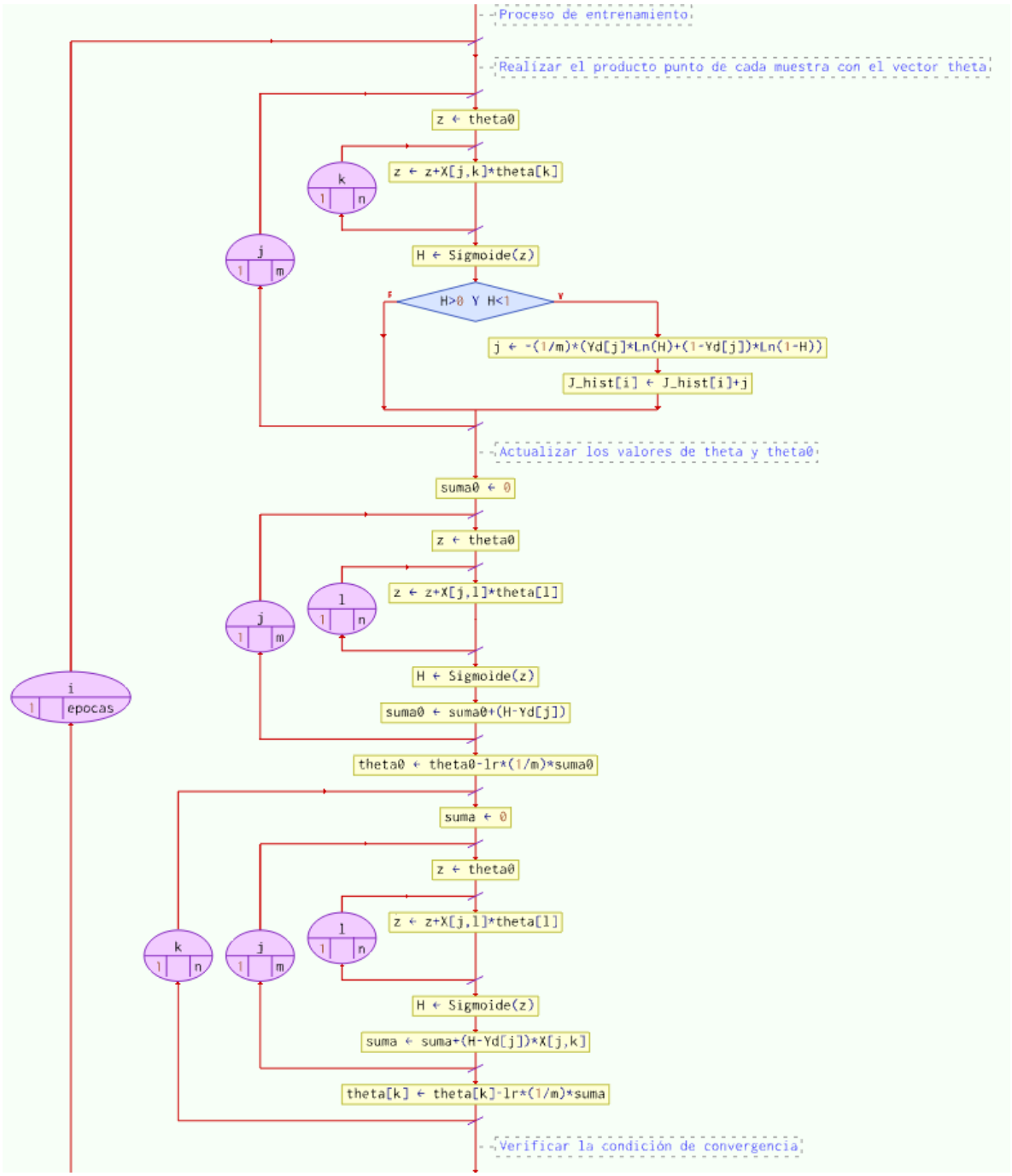


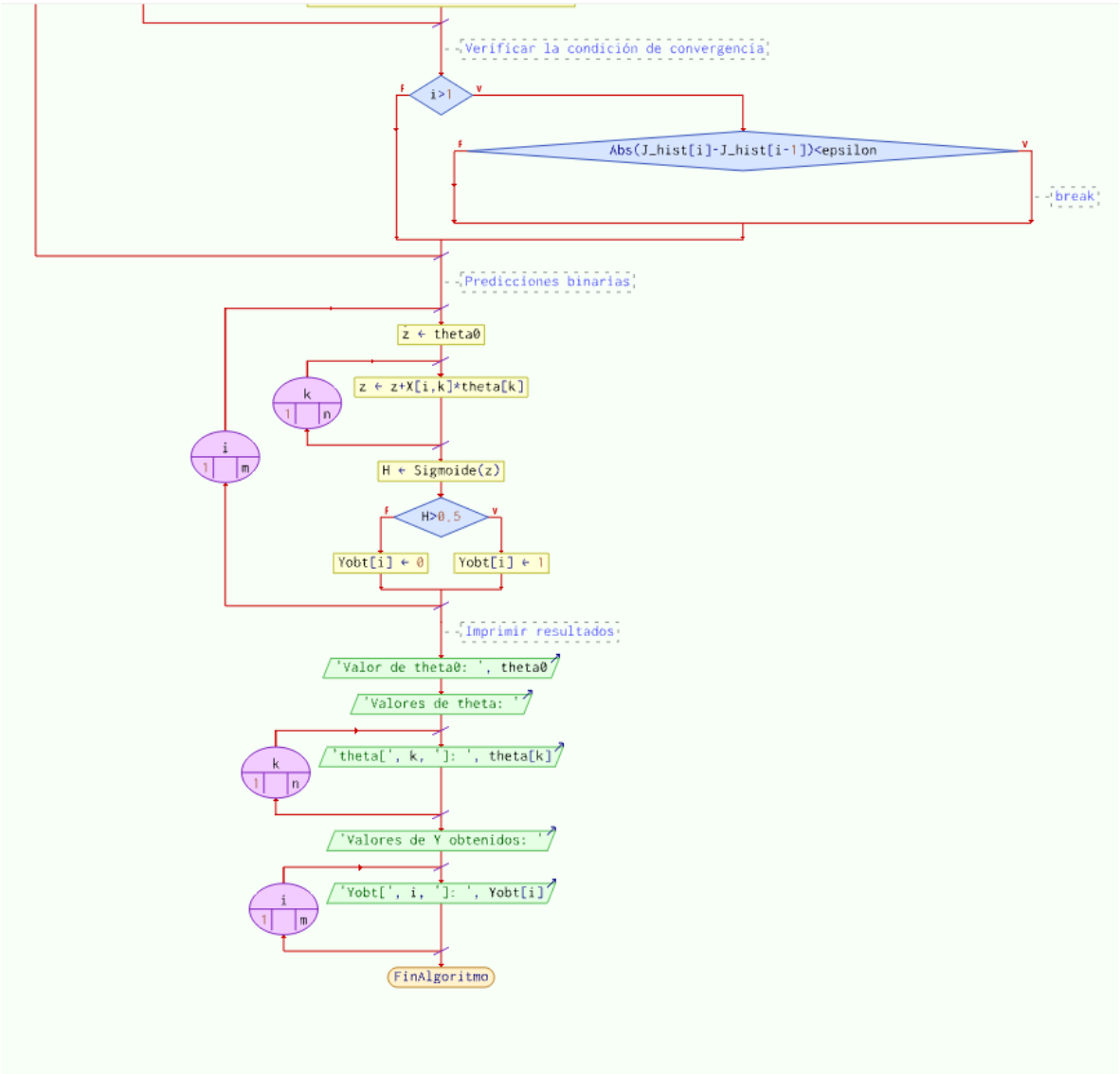
Figura 1.13. Pruebas de regresión logística. Fuente: Elaboración propia.

2.2.3. Ejercicio C

a) Diagrama de flujo




Proceso de entrenamiento
Realizar el producto punto de cada muestra con el vector theta
z ← theta0
k
1
n
z ← z+X[j,k]*theta[k]
j
1
m
H ← Sigmoide(z)
H>0 Y H<1
F
V
j ← -(1/m)*(Yd[j]*Ln(H)+(1-Yd[j])*Ln(1-H))
J_hist[i] ← J_hist[i]+j
Actualizar los valores de theta y theta0
suma0 ← 0
z ← theta0
l
1
n
z ← z+X[j,l]*theta[l]
j
1
m
H ← Sigmoide(z)
i
1
epocas
suma0 ← suma0+(H-Yd[j])
theta0 ← theta0-lr*(1/m)*suma0
suma ← 0
z ← theta0
l
1
n
z ← z+X[j,l]*theta[l]
k
1
n
j
1
m
H ← Sigmoide(z)
suma ← suma+(H-Yd[j])*X[j,k]
theta[k] ← theta[k]-lr*(1/m)*suma
Verificar la condición de convergencia



b) Código

```
import numpy as np
import matplotlib.pyplot as plt

epsilon = 0.01

# Definir la función sigmoide
def Sigmoide(z):
   return 1 / (1 + np.exp(-z))

def ValidateH(H):
   Predictions = np.zeros(len(H))
   for i in range(len(H)):
       if H[i] > 0.5:
           Predictions[i] = 1
       else:
```

```
            Predictions[i] = 0
    return Predictions

# Datos de entrenamiento
X =
np.array([[1,2],[1,3],[1,4],[1,5],[1,6],[1,7],[1,8],[1,9],[1,10],[1,11],[
1,12],[1,13],[1,14],[1,15],[1,16],[1,17],[1,18],[1,19],[1,20],[1,21]])
Yd = np.array([0,0,1,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,0])
Yobt = np.zeros(len(Yd))

# Definiendo hiperparámetros
n = len(X[0])
theta = np.zeros(n)
theta0 = 0

# Hiperparámetros dentro del modelo
lr = 0.1
epocas = 1000
m = len(Yd)

# Array para almacenar los valores de J
J_hist = np.zeros(epocas)

for i in range(epocas):
    Z = theta0 + np.dot(X, theta)
    H = Sigmoide(Z)

    # Evaluar la función de costo
    J = -(1/m) * np.sum(Yd * np.log(H) + (1 - Yd) * np.log(1 - H))

    J_hist[i] = J

    theta0 = theta0 - lr * (1/m) * np.sum(H - Yd)
    theta = theta - lr * (1/m) * np.dot(X.T, (H - Yd))

    if abs(J_hist[i] - J_hist[i-1]) < epsilon:
        break

Yobt = ValidateH(H)

print(f"Theta0: {theta0}")
print(f"Theta: {theta}")

plt.scatter(X[:, 1], Yd, color='blue', label='Datos de entrenamiento')

# Escalar la función de costo para que sea visible junto con los puntos
J_hist_scaled = (J_hist - np.min(J_hist)) / (np.max(J_hist) -
np.min(J_hist))
plt.plot(np.linspace(2, 21, epocas), J_hist_scaled, color='green',
label='Función de costo J (escalada)')
```

```
plt.xlabel('X')
plt.ylabel('Yd / H')
plt.title('Regresión logística')
plt.legend()
plt.show()
```

c) Funcionamiento

Para el caso de este ultimo ejercicio implementamos una forma de que programa no ejecute todas las épocas con las que fue inicializado sino que permita comparar un cierto rango entre J[i] y J[i-1] y así salir del bucle en cuanto minimice la diferencia entre este valor y épsilon. Dichas pruebas se muestran en la Figura 1.14.

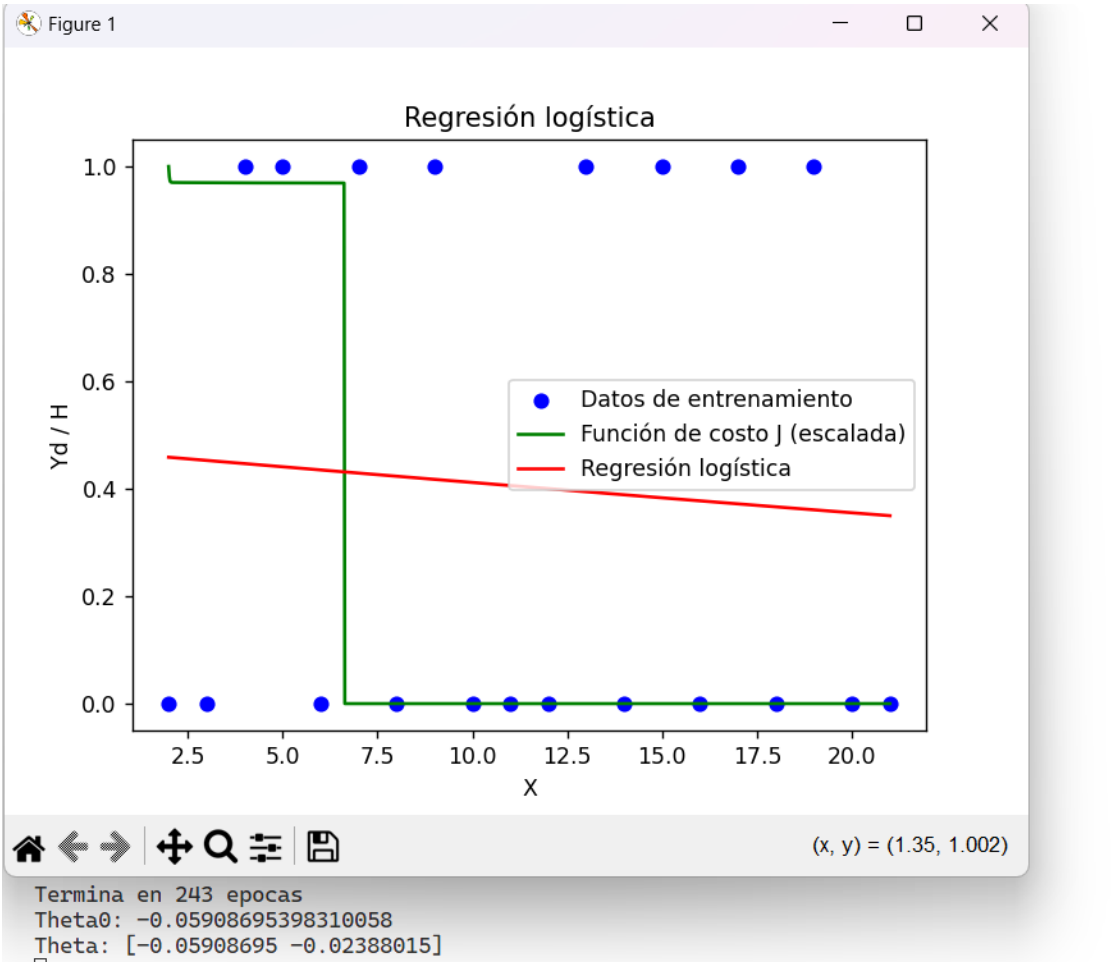


Figura 1.14. Pruebas de regresión logística con limitación de épocas. Fuente: Elaboración propia.

2.3. CONCLUSIÓN

En la práctica se utilizó la regresión logística donde se vio la forma de obtener las ecuaciones para la actualización de Θ_0 y Θ_1 a partir de la función que no tiene contemplado a Θ_0 para posteriormente ser utilizada en la parte 3 de la práctica. En el ejercicio B se solicitaron 15 datos y mostrar los valores de Θ_0 y Θ_1 y graficar J. dado que ya habías obtenido las fórmulas de actualización de Θ_0 y Θ_1, mostrar los valores de estas dentro del modelo dio valores muy pequeños como se muestra en la figura 1.15

```
Theta0: 0.09302713415611238
Theta: [ 0.09302713 -0.00718114]
```

Figura 1.15 Valores de Θ_0 y Θ_1. Fuente: Elaboración propia

Mostrando dando como resultado la figura 1.16 para la gráfica de J, el learning que se utilizo para la grafica fue de 0.01 dado que si se maneja un learningrate mas alto se mostraba como un rectángulo que cubría a casi su totalidad la gráfica. Como se muestra en la figura 1.17

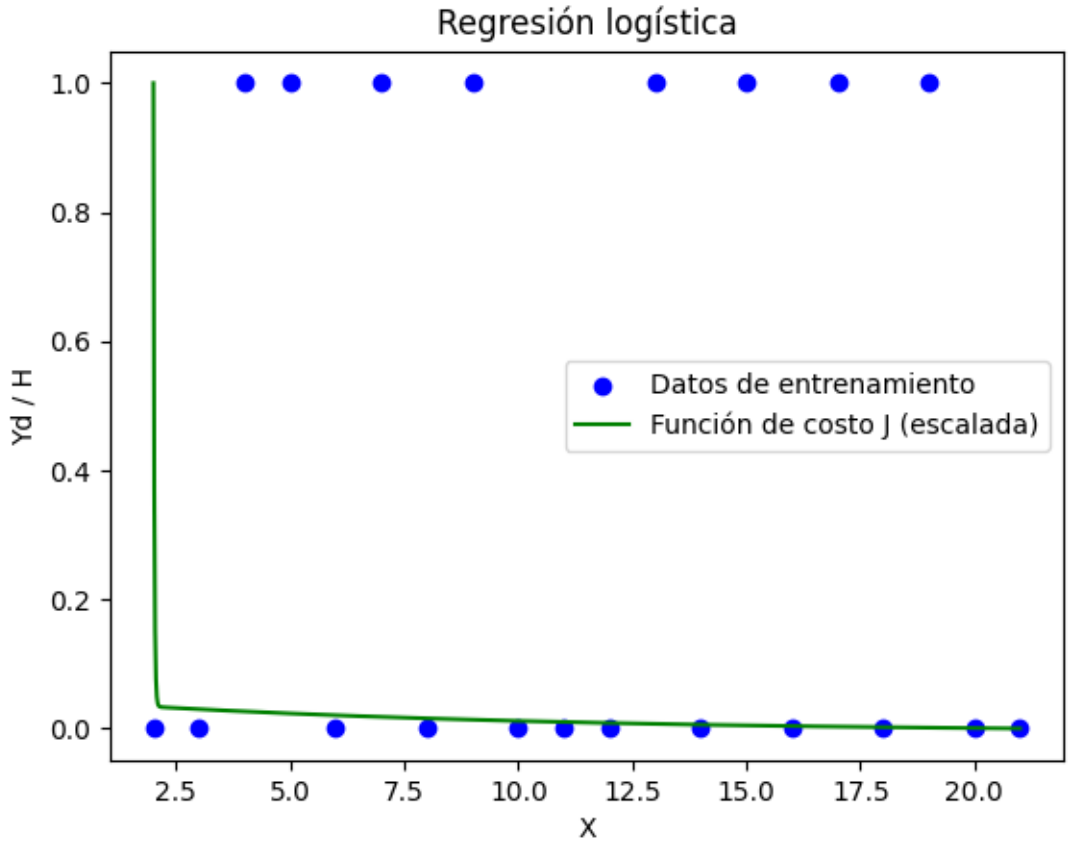


Figura 1.17 Grafica de J con los puntos clasificados y learningrate de 0.01. Fuente: Elaboración propia

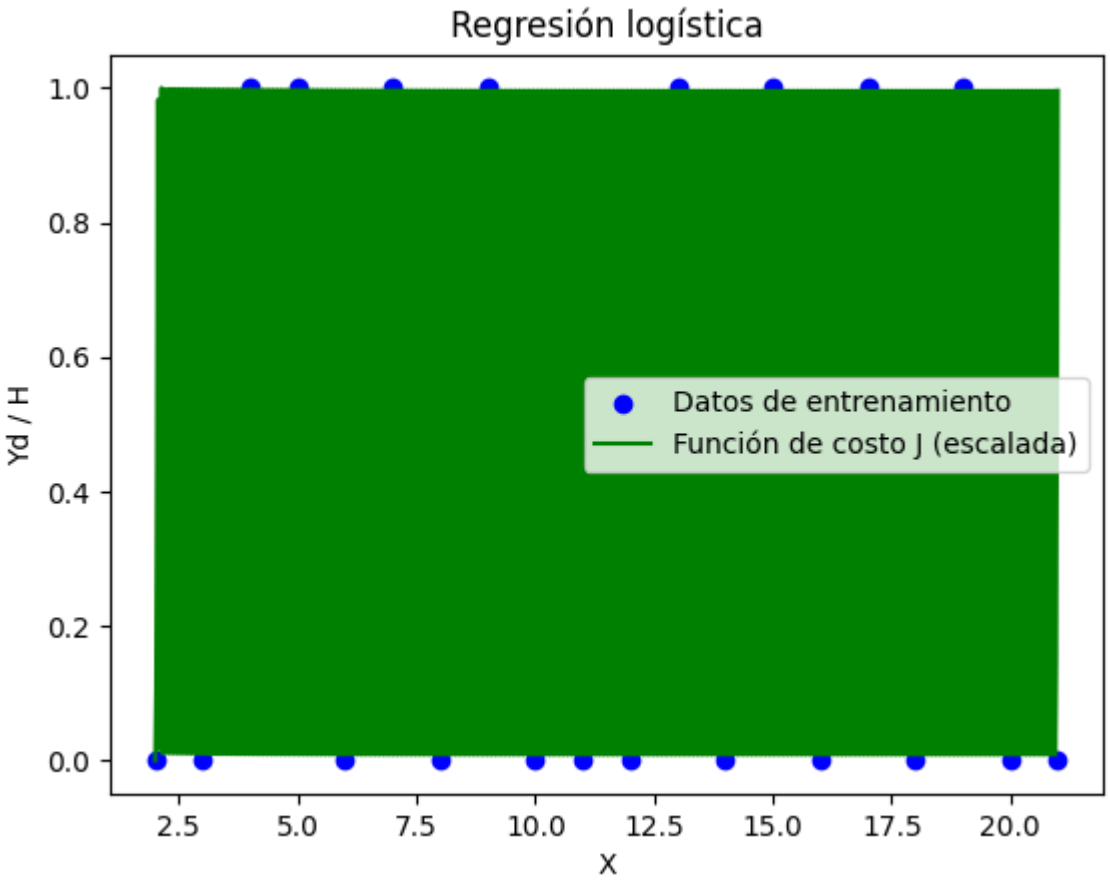


Figura 1.18 Grafica de J con los puntos clasificados y learningrate de 0.1. Fuente: Elaboración propia