Instituto Tecnológico y de Estudios Superiores de Monterrey

ESCUELA DE INGENIERÍA Y CIENCIAS

Inteligencia Artificial Avanzada para la Ciencia de Datos I

Manual de Usuario - Implementación Manual ML: Regresión Logística con Regularización.

Presenta:

Miguel Ángel Pérez Ávila

Profesor:

Dr. Víctor Manuel de la Cueva Hernández

Sante Fe, Ciudad de México a 28 de Agosto del 2025

Índice

Introducción

El presente documento presenta la implementación y metodología de ejecución de un programa escrito en Python que realiza procesos afines al entrenamiento de una regresión logística regularizada calculada mediante el ajuste de los parámetros θ mediante la optimización del error utilizando el método de Gradiente Descendiente. Se utilizó la técnica de vectorización en la medida de lo posible para realizar los cálculos involucrados. Las funciones implementadas incluyen:

- La lectura de un archivo de entrada.
- El cálculo de la función sigmoidal para un valor z dado.
- La construcción de una matriz de dos variables (m, 2) a una de grado 6 (m,28) agregando la columna bias.
- La ejecución del gradiente descendente regularizado aplicado a vectores X, Y y θ dados, considerando un paso Alpha a través de una cantidad de N épocas/iteraciones.
- El cálculo de la función de costo regularizada J(θ) para un vector X, Y y θ dados.
- La graficación de la dispersión de 2 vectores variables X1 y X2 dados indicando su etiqueta Y según su categoría (1 o 0) junto con la función ajustada mediante un vector optimizado de θ dados.
- La predicción de un vector X de prueba utilizando el vector de θ optimizadas para obtener el resultado de la clasificación y su accuracy.

Implementación

Es importante mencionar que para le ejecución de este programa es necesario contar con las siguientes librerías en el entorno de ejecución y una versión de python igual o cercana a la utilizada para esta implementación siendo esta la 3.11.13:

- **numpy** (Manipulación de vectores y operaciones matemáticas en general).
- **matplotlib** (Graficación y visualización de datos).
- **chardet** (Detección de codificación para la lectura de los archivos de entrada).
- **sklearn** (Utilizado únicamente como alternativa a la construcción manual implementada del vector a grado 6 mediante una herramienta que proporciona esta librería, su uso es meramente opcional)

En caso de no contar con alguna de las anteriores se pueden instalar y/o actualizar en bash mediante:

```
// En caso de utilizar conda
user@pc: conda activate <dev_env>

user@pc: pip install numpy matplotlib chardet scikit-learn
```

Se utilizó un enfoque orientado a objetos para la implementación de la regresión lineal en su totalidad, creando una estructura/clase llamada MLDev que proporciona las herramientas necesarias para realizar el cálculo de la regresión logística.

MLDev implementa la siguiente estructura explicada en términos de tipo de dato abstracto:

- Atributos de la estructura:
    - **X** → Matriz de características.
    - **Y** → Vector de etiquetas.
    - **Costos** → Historial de costos en gradiente descendente.

- Interfaz de la estructura:
    - Carga y preparación de datos:
        - **getFileEncoding(path)** → Detecta la codificación del archivo de datos.
        - **readFile(path, separador)** → Lee el dataset, construye las matrices X (features con bias incluido) y Y (etiquetas).
        - **setXVector(X)** → Define manualmente el vector/matriz de características.
        - **setYVector(Y)** → Define manualmente el vector de etiquetas.
        - **manualbuildPolynomialX(X)** →Método para extender una matriz con 2 predictores (X) a grado 6. La matriz de predictores X no se pasa con la columna de bias integrada.
        - **buildPolynomialX(degree, X, include_bias)** →Método para extender una matriz con n cantidad de predictores (X) a un grado deseado mediante la herramienta de scikit-learn. La matriz de predictores X no se pasa con la columna de bias integrada, más bien se llama al método con el parámetro include_bias = True.

    - Entrenamiento de modelo de regresión Logística:
        - **gradDescRegLog(n_epochs, thetas, X, y, alpha, lmbda = 1, regulate=True)**→ Calcula los parámetros óptimos mediante gradiente descendente con opción de regularización.
        - **calcCostRegLog(theta, X, y, lmbda = 1, regulate=True)** → Calcula la función de costo J(θ) con opción de regularización.
        - **calcGradRegLog(theta, X, y)** → Método para el cálculo vectorizado de la gradiente dados los vectores X, Y y θ.

    - Visualización:
        - **graph(x, y, xlabel, ylabel, title)** → Gráfica de curvas como evolución de error vs. épocas.
        - **scatterPlotAndRegression6Degree(x, y, thetas)** → Gráfica de dispersión de datos y la función polinomial ajustada.

Mediante la estructura anterior se implementan las siguientes funciones utilizadas para este proyecto:

```
# Función para graficar la dispersión de datos (Dados los Vectores X y Y) y la función ajustada
# Está función ASUME que el vector de X aún contiene la columna X0 del bias con "unos"
# - Input:
#       - x: Vector de X
#       - y: Vector de Y
```

```
#       - theta: Vector de thetas óptimas p
# - Return : Void
def graficaDatos(x, y, theta)

# Función para optimizar y conseguir el vector de thetas óptimo que ajuste a los vectores de datos dados a razón
# de un paso (alpha) definido y una n cantidad de iteraciones
# - Input:
#       - theta: Vector de thetas iniciales para el entrenamiento
#       - x: Vector de X
#       - y: Vector de Y
#       - iteraciones: épocas a ejecutar
#       - alpha: Learning Rate o paso para el ajuste de pesos
#       - lmbda: constante de regularización
# - Return :
#       - thetas: np.array de thetas óptimas
def aprende(theta, X, y, iteraciones, alpha = 0.01, lmbda = 1)

# Función para realizar predicciones utilizando las thetas óptimas del modelo
# y la matriz de X
# - Input:
#       - theta: Vector de thetas a evaluar
#       - x: Vector de X
#       - y: Vector de Y
#       - lmbda: constante de regularización
# - Return :
#       - Y_predicted
def predice(theta, X, umbral = 0.5)

# Función para calcular el costo dado un Vector de X, Y y thetas y la gradiente correspondiente
# - Input:
#       - theta: Vector de thetas a evaluar
#       - x: Vector de X
#       - y: Vector de Y
#       - lmbda: constante de regularización
# - Return :
#       - Costo
#       - Gradiente
def funcionCostoReg(theta, X, y , lmbda)

# Función para calcular el valor de la función sigmoide para un valor z
# - Input:
#       - z
# - Return :
#       - 1/(1+np.exp(-z))
def sigmoidal(z)

# Método para extender una matriz con 2 predictores (X) a grado 6
# La matriz de predictores X no se pasa con la columna de bias integrada, todo se construye en esta función
# - Input:
#       - x: Vector de X
# - Return :
#       - mappedmatrix
def mapeoCaracteristicas(X, tipo="Manual")
```

Ejemplos de Ejecución

A continuación se muestra un ejemplo de ejecución para describir la manera en la que se deben utilizar las funciones anteriormente mencionadas utilizando los datos de prueba brindados:

- En caso de querer leer un archivo con las entradas para construir los vectores se puede hacer uso del método readFile de la clase MLDev que recibe la ruta del archivo asumiendo que no contiene encabezados para las columnas y que tiene un carácter con el cual se realiza la separación e interpretación del dataset.

```
mlObject = MLDev() # Instancia para el manejo de procesos de ML
path = "ex1data1.txt"

# Lectura de archivo mediante un carácter separador considerando que no hay encabezados para las columnas y sin agregar la columna de "unos" o bias
# y posteriormente se realiza el llenado de vectores
mlObject.readFile(path, ",", omitColumnTitles=False, addX0Col=False)
```

Para acceder a los vectores construidos se pueden utilizar las propiedades de la clase que contienen los arreglos de X y de Y. Para construir la matriz X al grado 6 se realiza lo siguiente:

```
mlObject.X # Vector con X incluyendo x0
mlObject.Y # Vector con Etiquetas Y

# Construcción de matriz (118,2) a grado 6 (118,28) utilizando el método manual de construcción
 mlObject.X = mapeoCaracteristicas(mlObject.X)
```

- **Función “aprende”:** Para utilizar la función que calcula el gradiente descendente regularizado es necesario contar con un vector arreglado de X, Y y thetas (En caso de haber construido los vectores con el método readFile anteriormente mencionado se pueden utilizar directamente los atributos que almacenan estos vectores en la estructura). Se debe indicar un paso alpha (0.01 utilizado como estándar predeterminado) y un número definido de iteraciones/épocas durante las cuales se realizará la optimización. Al final la función retorna el vector thetas con los pesos actualizados y óptimos, además muestra una gráfica donde se muestra el costo a través de las épocas del entrenamiento para evaluar el aprendizaje del modelo. Para el ejemplo se utiliza un vector de thetas inicializado con valores “0” y un valor de lambda para la regularización de “1”:

```
# Vector de thetas iniciales tamaño (n_variables, 1)
thetas = np.zeros((mlObject.X.shape[1], 1))
print("\nThetas Iniciales: \n", thetas.T)

# Constante de regularización
LMBDA = 1

# Thetas óptimas resultantes del Entrenamiento en busca de thetas óptimas mediante gradiente descendente
thetas = aprende(thetas, mlObject.X, mlObject.Y, 20000, alpha = 0.01, lmbda=LMBDA)
```

```
print("\nThetas Final    : \n", thetas)
```

Mostrando como output:

Thetas Final   :

[[1.20521812]

[0.58259575]

[1.13134301]

[-1.92013423]

[-0.83056988]

[-1.28496215]

[0.10189608]

[-0.34252552]

[-0.34135251]

[-0.18125994]

[-1.40956855]

[-0.06604411]

[-0.58032641]

[-0.24610054]

[-1.14672102]

[-0.24322098]

[-0.20113273]

[-0.05872165]

[-0.25849166]

[-0.27167027]

[-0.49102021]

[-1.01576224]

[0.0110959]

[-0.28016002]

[0.00479444]

[-0.309752]

[-0.12602254]

[-0.94014043]]

El gráfico de costo a través de las épocas se visualiza de la siguiente manera:

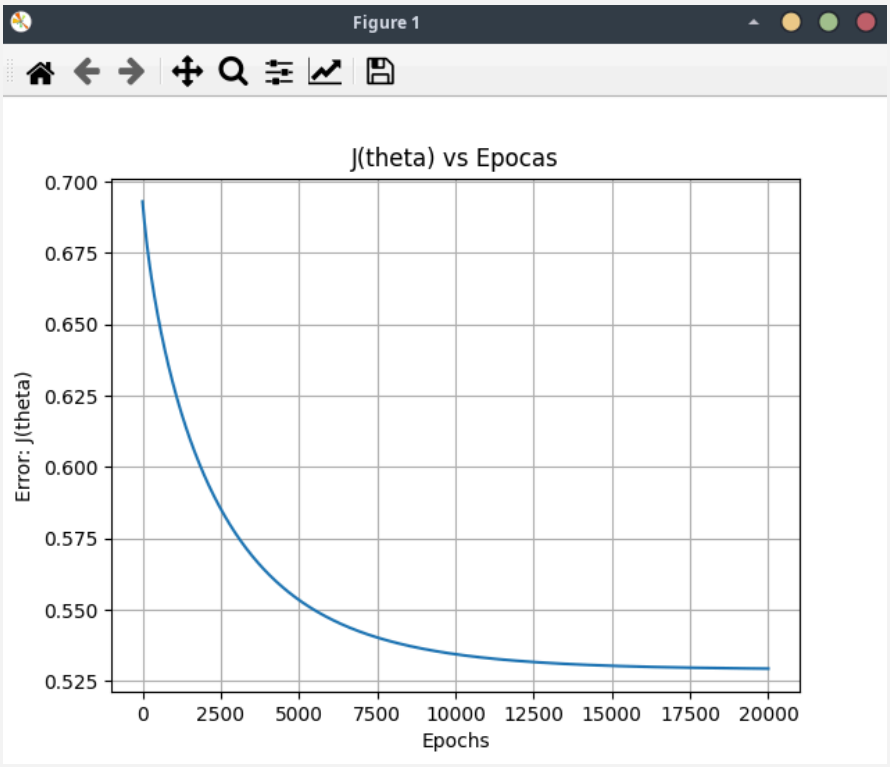

Figura 1: Ejemplo de visualización gráfica del costo a través de las épocas.

Para realizar una predicción con los valores thetas óptimos obtenidos se puede realizar de la siguiente manera utilizando la función “predice” y evaluar el accuracy del modelo:

```
# Ejemplo de Predicciones
p = predice(thetas, mlObject.X)
# Contador de predicciones correctas
 correctos = 0
 for i in range(len(p)):
    if p[i][0] == mlObject.Y[i][0]:
        # Correcto
        correctos+=1
```

```
# Accuracy del modelo
print("Accuracy: ",correctos/len(p)*100, "%")
```

Mostrando el siguiente output:

Accuracy: 81.35593220338984 %

- **graficaDatos:** Para utilizar la función que grafica la dispersión de los vectores de entrada X y Y, para además dibujar la función ajustada con los valores óptimos resultantes para theta, es necesario colocar como parámetros el arreglo de X (Asumiendo que aún contiene la columna de x0 con "unos"), el arreglo de Y (De dimensiones (m, 1)) y el vector de thetas con dimensión de grado 6 (28, 1). La graficación se ejecuta de la siguiente manera:

```
# Graficación de dispersión con función ajustada visible
graficaDatos(mlObject.X, mlObject.Y, thetas)
```

La gráfica se vería de la siguiente manera:

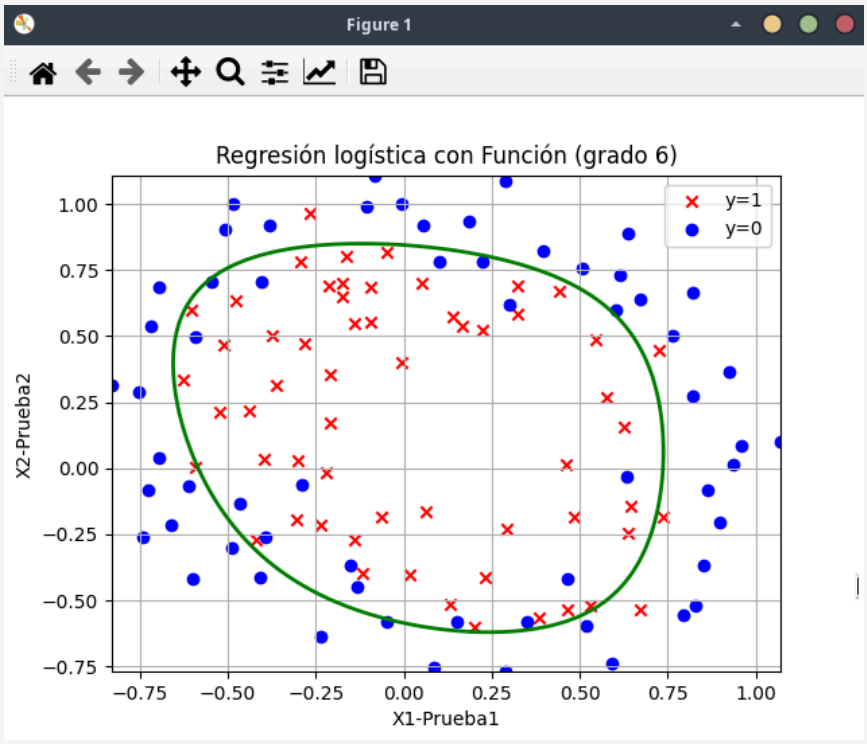

Figura 2: Ejemplo de visualización gráfica de dispersión de los datos y la función ajustada mediante la regresión mostrando la frontera de decisión.

- **calcularCosto:** Para utilizar la función que calcula el costo regularizado, se debe colocar como parámetros un vector construido de X, un vector Y y un vector de thetas respecto a las cuales se calculará el costo utilizando un parámetro para lambda definido. Lo anterior se demuestra en el siguiente ejemplo:

```
# Ejemplo de Cálculo de Costo para Thetas iniciales
J, grad = funcionCostoReg(thetas, mlObject.X, mlObject.Y, lmbda=LMBDA)
print("\nError para thetas Iniciales: ",J)
```

El output se puede visualizar de la siguiente manera:

Ejemplo de cálculo de costo:

Error para thetas Iniciales: 0.6931471805599454

A continuación se muestran los efectos producidos por la variación de la constante lambda para 0, 1 y 100 con 20000 épocas de entrenamiento:

Prueba - Lambda = 0

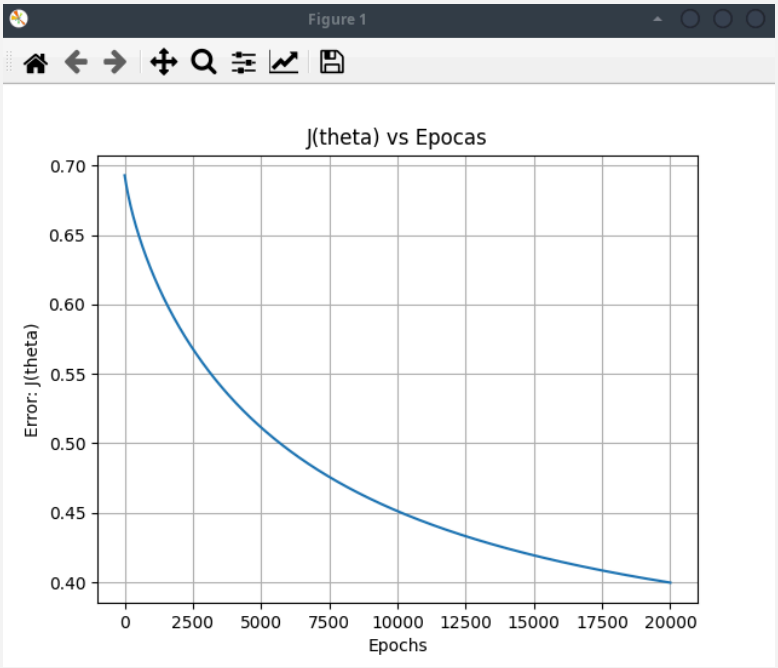

Figura 3: Ejemplo de visualización gráfica del costo a través de las épocas. Lambda=0

Thetas Final :

[[1.87513654]

[1.18538765]

[2.02333963]

[-3.10622974]

[-1.68980309]

[-2.13298265]

[0.32233717]

[-0.67451577]

[-0.63347263]

[-0.23971435]

[-2.37398707]

[-0.14002374]

[-1.0225326]

[-0.53686275]

[-1.94045025]

[-0.35697046]

[-0.38710541]

[-0.07578873]

[-0.48704287]

[-0.54678907]

[-0.75760525]

[-1.74574934]

[0.01561567]

[-0.4927096]

[0.00557678]

[-0.55737576]

[-0.2978593]

[-1.5640682]]

Accuracy: 81.35593220338984 %

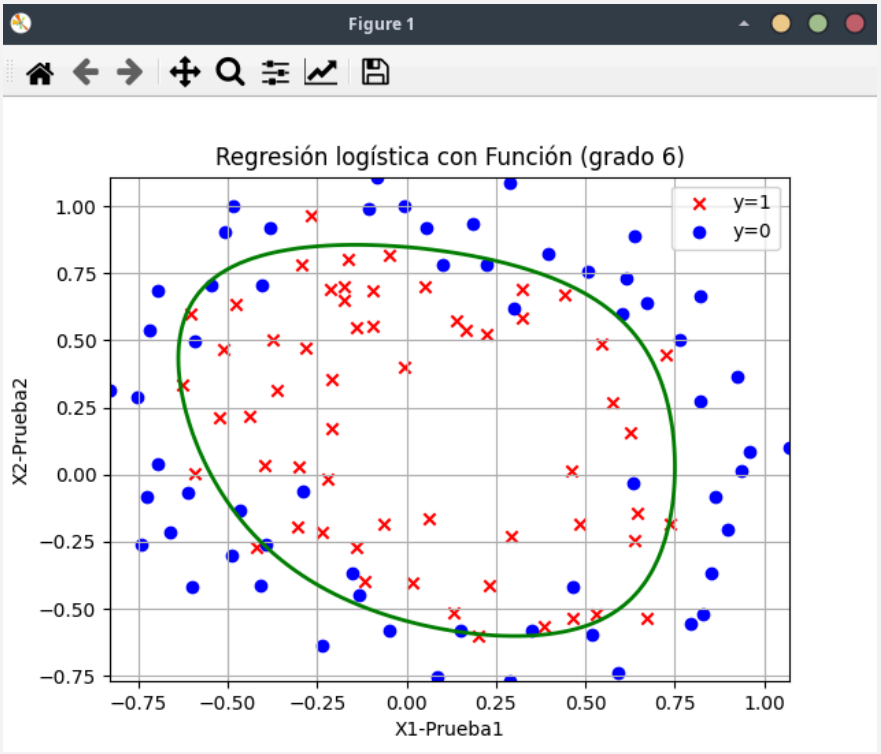

Figura 4: Ejemplo de visualización gráfica de dispersión de los datos y la función ajustada mediante la regresión mostrando la frontera de decisión para lambda 0.

Prueba - Lambda = 1

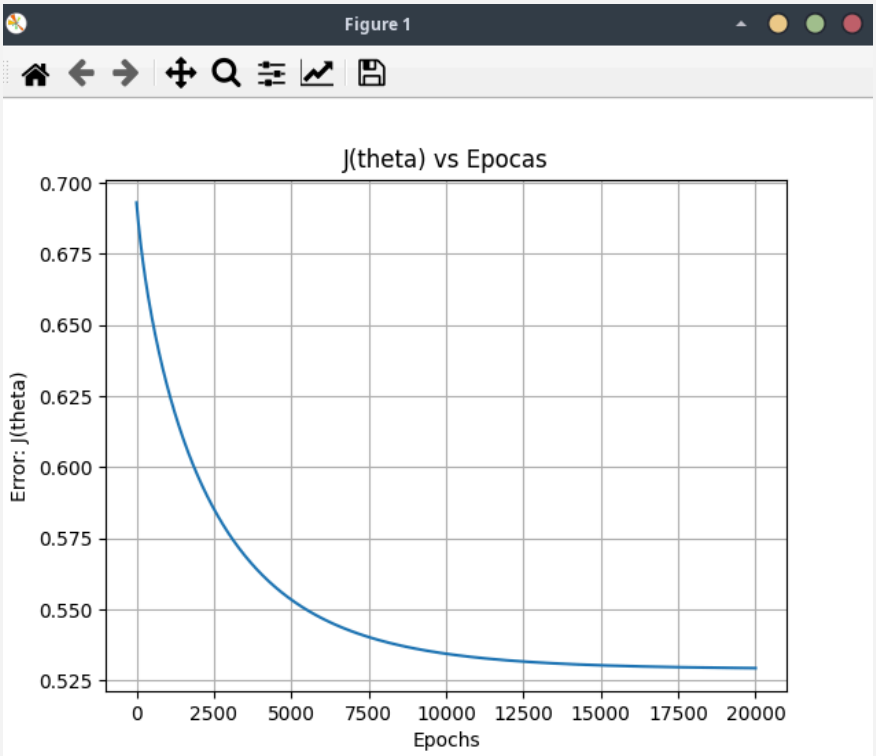

Figura 5: Ejemplo de visualización gráfica del costo a través de las épocas. Lambda=1

Thetas Final :

[[1.20521812]

[0.58259575]

[1.13134301]

[-1.92013423]

[-0.83056988]

[-1.28496215]

[0.10189608]

[-0.34252552]

[-0.34135251]

[-0.18125994]

[-1.40956855]

[-0.06604411]

[-0.58032641]

[-0.24610054]

[-1.14672102]

[-0.24322098]

[-0.20113273]

[-0.05872165]

[-0.25849166]

[-0.27167027]

[-0.49102021]

[-1.01576224]

[0.0110959]

[-0.28016002]

[0.00479444]

[-0.309752]

[-0.12602254]

[-0.94014043]]

Accuracy: 83.05084745762711 %

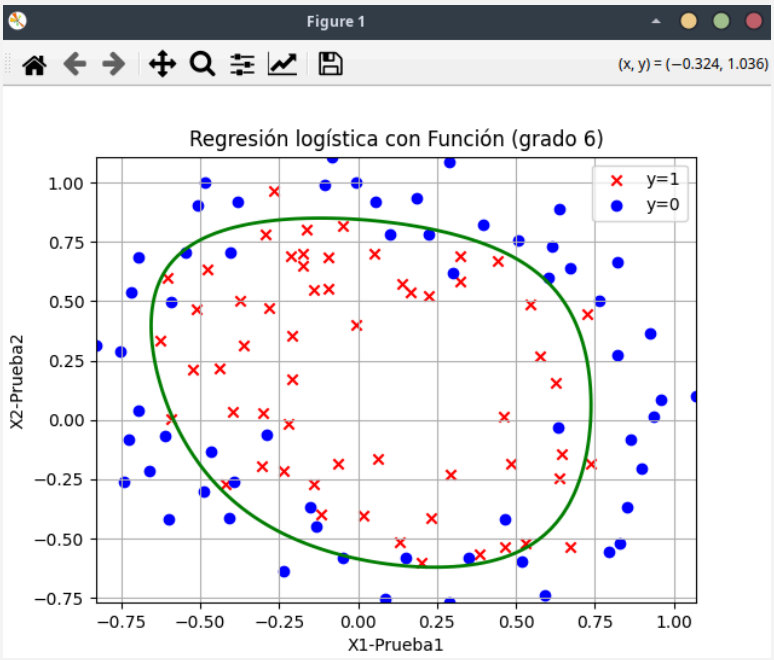

Figura 6: Ejemplo de visualización gráfica de dispersión de los datos y la función ajustada mediante la regresión mostrando la frontera de decisión para lambda 1.

Prueba - Lambda = 100

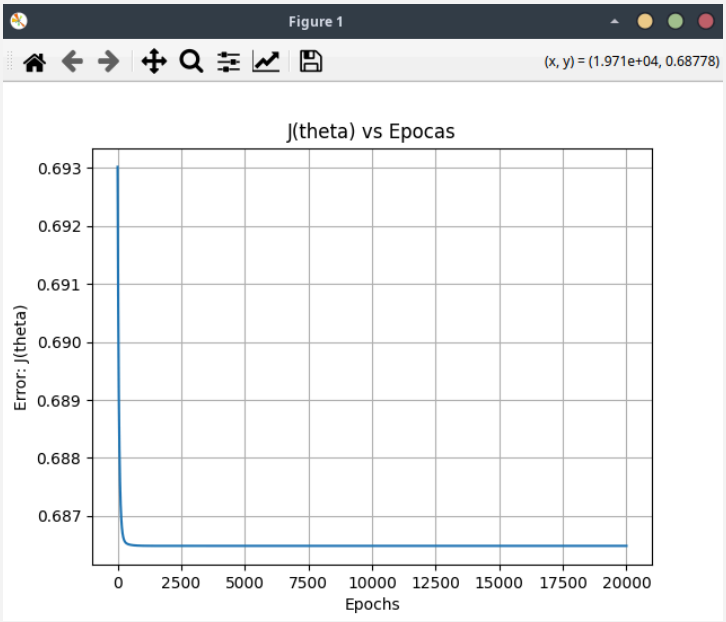

Figura 7: Ejemplo de visualización gráfica del costo a través de las épocas. Lambda=100

Thetas Final :

[[0.02187771]

[-0.01748172]

[0.00571079]

[-0.05516895]

[-0.01314877]

[-0.03859858]

[-0.01846356]

[-0.00773219]

[-0.00892429]

[-0.02280452]

[-0.04343846]

[-0.00235623]

[-0.01415612]

[-0.00349508]

[-0.04143588]

[-0.02100593]

[-0.00471917]

[-0.00359131]

[-0.00632226]

[-0.00502441]

[-0.03197676]

[-0.03416335]

[-0.00107629]

[-0.00702615]

[-0.00038506]

[-0.0079823]

[-0.00154779]

[-0.04108677]]

Accuracy: 61.016949152542374 %

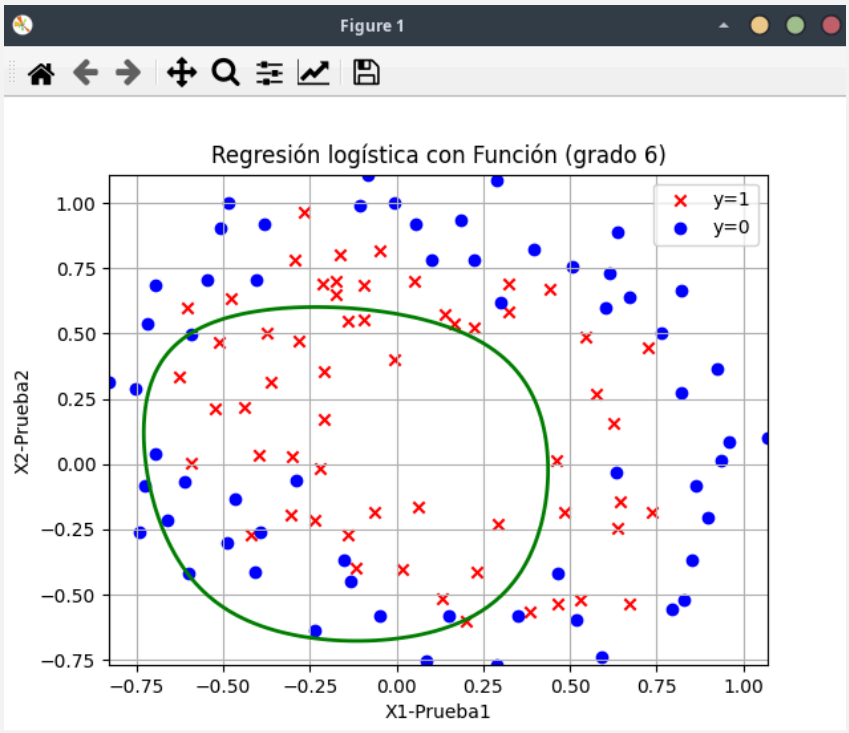

Figura 8: Ejemplo de visualización gráfica de dispersión de los datos y la función ajustada mediante la regresión mostrando la frontera de decisión para lambda 100.

Conclusión

Se puede observar claramente que el efecto de la regularización para los distintos valores de lambda es sumamente significativo. El resultado de una regularización exagerada como lambda=100 donde los valores del vector de pesos son todos enormemente reducidos a 0, lo cual le quita de forma general el efecto que cada una de estas variables produce en el resultado final, además del accuracy obtenido de 61.02% que es muy bajo para el modelo. En general es claro el concepto que se visualiza en este modelo siendo underfitting lo que se observa. Posteriormente, con un valor de lambda=1, se puede observar que obtenemos un accuracy de 83.05% y un vector de pesos balanceado en sus valores, de los cuales se puede identificar algunas variables que se logran reducir moderadamente hacia 0 lo cuál apoyará a la reducción del grado propuesto, en general es un buen modelo de acuerdo con estas métricas. Finalmente para un valor de lambda=0 se puede observar en la gráfica de la evolución del costo a través de las épocas ejecutadas como se presenta el fenómeno conocido como overfitting, siendo que el modelo se pretende ajustar de sobremanera a los datos, sin embargo para datos ajenos al conjunto de entrenamiento se puede esperar un bajo rendimiento del modelo para las posibles predicciones resultantes.

En general se comprueba el hecho de que el hiper parámetro lambda para la regularización no tiene un valor comprobado para brindarle precisión a un modelo, más bien la prueba y error es la técnica adecuada para que el científico de datos evalúe su efecto en el conjunto de datos utilizado para cada situación específica.

Finalmente, se puede decir que mediante la utilización de los procesos implementados anteriormente para ejecutar el entrenamiento de un modelo de regresión logística utilizando la técnica de gradiente descendente regularizada para encontrar los valores óptimos de los pesos para una función de grado 6 propuesta, es posible realizar la predicción del funcionamiento de los chips involucrados en la problemática utilizando los valores óptimos de theta obtenidos por el modelo y una constante lambda de 1:

Thetas Final :

[[1.20521812]

[0.58259575]

[1.13134301]

[-1.92013423]

[-0.83056988]

[-1.28496215]

[0.10189608]

[-0.34252552]

[-0.34135251]

[-0.18125994]

[-1.40956855]

[-0.06604411]

[-0.58032641]

[-0.24610054]

[-1.14672102]

[-0.24322098]

[-0.20113273]

[-0.05872165]

[-0.25849166]

[-0.27167027]

[-0.49102021]

[-1.01576224]

[0.0110959]

[-0.28016002]

[0.00479444]

[-0.309752]

[-0.12602254]

[-0.94014043]]

Accuracy: 83.05084745762711 %