

# LABORATORIO 03

## MATERIA: INTELIGENCIA ARTIFICIAL

ESTUDIANTE: Alvarez Alain Jonathan

**AÑO:2024**

Carrera: Ing. en Diseño y Animación Digital

### 1.- RESUMEN EXPLICANDO LO TRABAJADO

Lo primero que hicimos fue importar lo que serían las librerías que usaríamos, mas que nada la librería de Pandas y enlazar nuestro drive



The screenshot shows a Google Colab notebook interface. At the top, the notebook is titled 'z\_Alain\_Jonathan.ipynb'. Below the title bar, there are tabs for 'Entorno de ejecución', 'Herramientas', and 'Ayuda', along with a status message 'Se han guardado todos los cambios'. The main area of the notebook displays two code cells. The first cell, which has been executed successfully (indicated by a green checkmark and '0 s'), contains Python code for importing various libraries: 'os' for file handling, 'numpy' for scientific computing, 'pyplot' from 'matplotlib' for plotting, 'optimize' from 'scipy' for optimization, and 'pandas' for data manipulation. It also includes a magic command '%matplotlib inline' to display plots within the notebook. The second cell, also executed successfully (green checkmark and '1 min'), shows the code to mount the Google Drive using 'google.colab.drive'. Below this code, a message indicates that the drive is mounted at the path '/content/gdrive'.

```
# Se utiliza para el manejo de rutas y directorios.
import os

# Calculo científico y vectorial para Python
import numpy as np

# Librerías para graficar
from matplotlib import pyplot as plt

# Módulo de optimización de scipy
from scipy import optimize

# Le dice a matplotlib que incruste gráficos en el cuaderno
%matplotlib inline

import pandas as pd

from google.colab import drive
drive.mount("/content/gdrive")

Mounted at /content/gdrive
```

Luego importamos lo que sería nuestro dataset seleccionado en clases, de este escogemos nuestras X y nuestra Y, de esto dividimos nuestros datos en 80% entrenamiento y el otro 20% prueba

Jonathan.ipynb ☆

ejecución Herramientas Ayuda Se han guardado todos los cambios

Comentario Comparti

+ Código + Texto

RAM Disco

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

# Cargar datos del archivo CSV limpio
data = pd.read_csv('/content/gdrive/MyDrive/binario_limpio.csv', delimiter=',') # Ajustar la ruta si es necesario

# Asignar X con las columnas 1, 2, 3, 4, 5, 6 y y con la columna 0
X = data.iloc[:, [1, 2, 3, 4, 5, 6]].values
y = data.iloc[:, 0].values

# Dividir los datos en entrenamiento (80%) y prueba (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Mostrar dimensiones de los conjuntos de datos
print("Dimensiones de X_train:", X_train.shape)
print("Dimensiones de X_test:", X_test.shape)
print("Dimensiones de y_train:", y_train.shape)
print("Dimensiones de y_test:", y_test.shape)
```

Dimensiones de X\_train: (202928, 6)  
Dimensiones de X\_test: (50733, 6)  
Dimensiones de y\_train: (202928,)  
Dimensiones de y\_test: (50733,)

## Graficamos nuestros datos

```
def plotData(X, y):
    # Gráfica los puntos de datos X y y en una nueva figura.
    # Grafica los puntos de datos con * para los positivos y o para los negativos.

    # Crea una nueva figura
    fig = plt.figure()

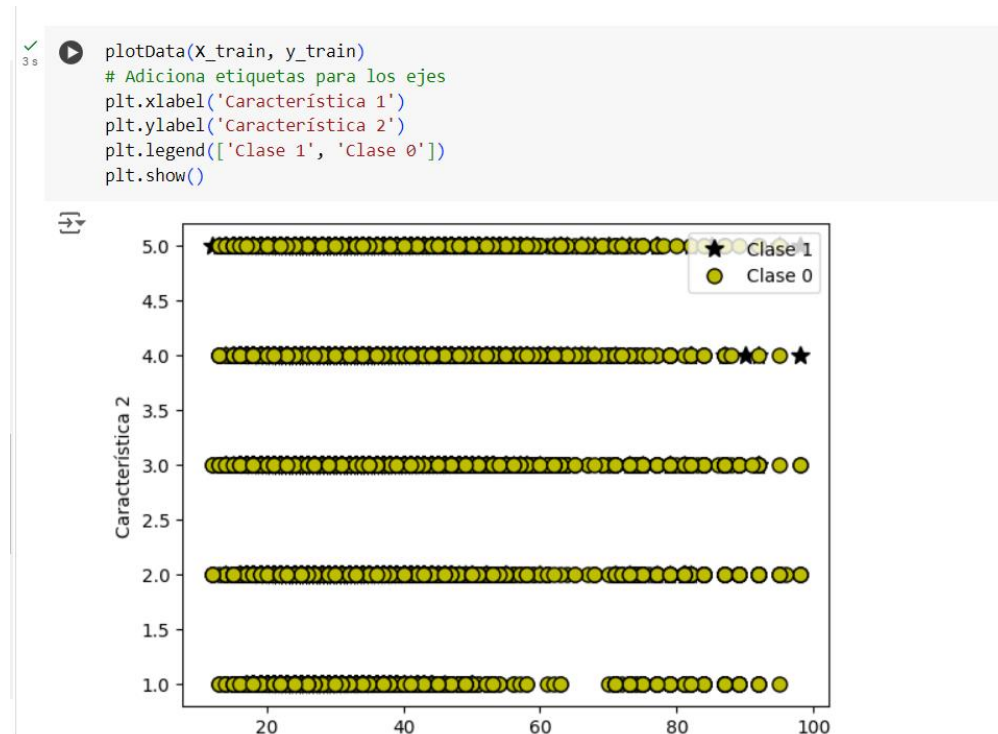
    # Encuentra índices de ejemplos positivos y negativos
    pos = y == 1
    neg = y == 0

    # Grafica ejemplos
    plt.plot(X[pos, 0], X[pos, 1], 'k*', lw=2, ms=10)
    plt.plot(X[neg, 0], X[neg, 1], 'ko', mfc='y', ms=8, mec='k', mew=1)
```

Se llama a la función implementada para mostrar los datos cargados:

```
[10] plotData(X_train, y_train)
# Adiciona etiquetas para los ejes
plt.xlabel('Característica 1')
plt.ylabel('Característica 2')
plt.legend(['Clase 1', 'Clase 0'])
plt.show()
```

## Mostramos la grafica



## Aplicamos las formulas aprendidas en clases

+ Código + Texto

✓ RAM  
Disco

◆ Gemini

^

Los resultados que debe generar la funcion sigmoidea para valores positivos amprios de x, deben ser cercanos a 1, mientras que para valores negativos grandes, la sigmoide debe generar valores cercanos 0. La evaluacion de `sigmoid(0)` debe dar un resultado exacto de 0.5. Esta funcion tambien debe poder trabajar con vectores y matrices.

```
✓ [11] def sigmoid(z):
      # Calcula la sigmoide de una entrada z
      z = np.array(z)
      g = 1 / (1 + np.exp(-z))
      return g
```

Se calcula el valor de la sigmoide aplicando la funcion sigmoid con `z=0`, se debe obtener un resultado de 0.5. RE recomienda experimentar con otros valores de `z`.

```
✓ 0 s [12] # Prueba la implementación de la función sigmoid
      z = [0, 0.5, 1]
      g = sigmoid(z)
      print('g( ', z, ' ) = ', g)
```

```
g( [0, 0.5, 1] ) = [0.5      0.62245933 0.73105858]
```

1.2.2 Función de Costo y Gradiente

Se implementa la funcion cost y gradient, para la regresión logística. Antes de continuar es importante agregar el termino de intercepcion a X.

```
✓ [13] # Configurar la matriz adecuadamente y agregar una columna de unos que corresponde al término de intercepción.
      m, n = X_train.shape
```

pacio disponible

Damos a Alpha un valor y damos un número determinado de interacciones y luego modificamos el X\_array según nuestro dataset

```
+ Código + Texto
[15] return theta, J_history

# Elegir algún valor para alpha (probar varias alternativas)
alpha = 0.001
num_iters = 100

# Inicializa theta y ejecuta el descenso por el gradiente
theta = np.zeros(n + 1)
theta, J_history = descensoGradiente(theta, X_train, y_train, alpha, num_iters)

# Gráfica la convergencia del costo
plt.plot(np.arange(len(J_history)), J_history, lw=2)
plt.xlabel('Número de iteraciones')
plt.ylabel('Costo J')

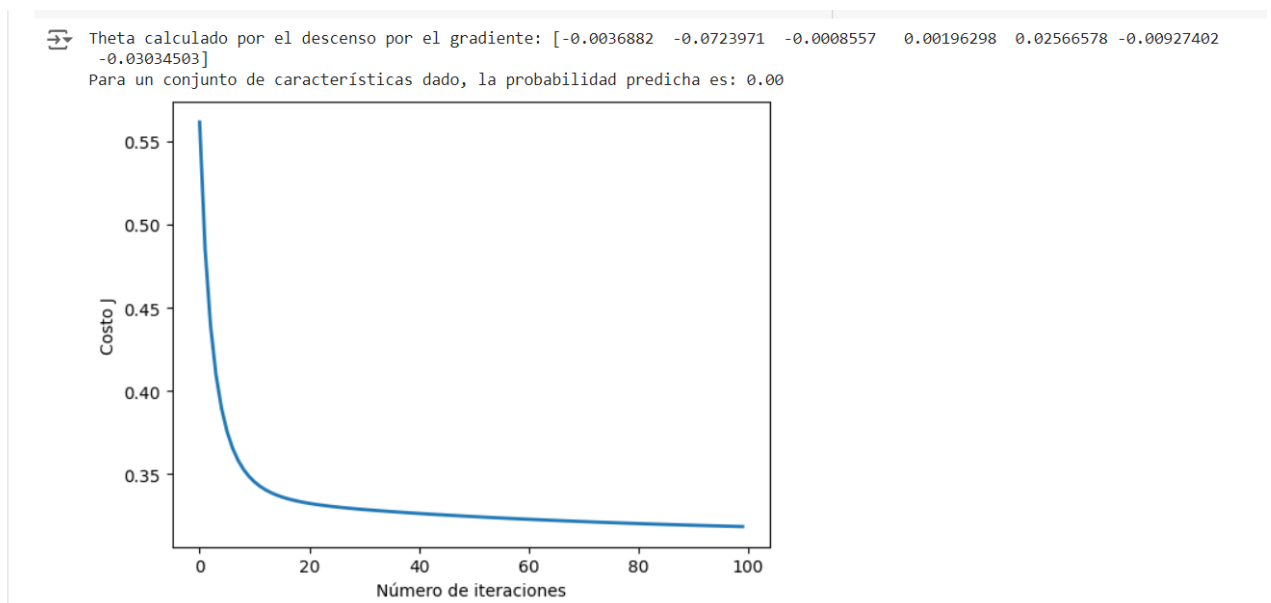
# Muestra los resultados del descenso por el gradiente
print('Theta calculado por el descenso por el gradiente:', theta)

# Verificar predicción para un conjunto específico
X_array = [1, 80, 46, 70, 60, 50, 85] # Ajustar según características
aprueba = sigmoid(np.dot(X_array, theta))

print('Para un conjunto de características dado, la probabilidad predicha es: {:.2f}'.format(aprueba))

Theta calculado por el descenso por el gradiente: [-0.0036882 -0.0723971 -0.0008557  0.00196298  0.02566578 -0.00927402
-0.03034503]
Para un conjunto de características dado, la probabilidad predicha es: 0.00
```

## Graficamos



En la siguiente formula cambiamos las variables de test\_theta , gradiante en theta inicial de acuerdo a nuestro dataset.

Se prueba la función `costFunction` utilizando dos casos de prueba para  $\theta$ .

```
[ ] # Inicialización de parámetros de ajuste
initial_theta = np.zeros(n + 1)
cost, grad = costFunction(initial_theta, X_train, y_train)

print('Costo en theta inicial (zeros): {:.3f}'.format(cost))
print('Gradiente en theta inicial (zeros):', grad)
```

```
Costo en theta inicial (zeros): 0.693
Gradiente en theta inicial (zeros): [ 0.4054098  11.40110532  0.93795583  1.1489617  1.25925205  3.05923776
 2.54001419]
```

```
[ ] # Calcula y muestra el costo y el gradiente con valores de theta diferentes a cero
test_theta = np.array([-24, 0.2, 0.2, 0.1, 0.3, 0.4, 0.5]) # Ajustar dimensiones si es necesario
cost, grad = costFunction(test_theta, X_train, y_train)

print('Costo en theta de prueba: {:.3f}'.format(cost))
print('Gradiente en theta de prueba:', grad)
```

```
Costo en theta de prueba: 0.773
Gradiente en theta de prueba: [-0.0658074 -1.72978066 -0.2005593 -0.01194694 -0.05307026 -0.69049167
-0.33246881]
```

Luego usamos los siguientes códigos para un parámetros de aprendizaje usando `scipy.optimize`

```
# Establecer las opciones para optimize.minimize
options = {'maxiter': 1000}

# Revisar la documentación de scipy's optimize.minimize para mayor descripción de los parámetros
res = optimize.minimize(costFunction,
                        initial_theta,
                        (X_train, y_train),
                        jac=True,
                        method='TNC',
                        options=options)

# La propiedad `fun` del objeto devuelto por `OptimizeResult` contiene el valor del costFunction de un theta optimizado
cost = res.fun

# Theta optimizada está en la propiedad `x`
theta = res.x

# Imprimir theta optimizada
print('Costo con un valor de theta encontrado por optimize.minimize: {:.3f}'.format(cost))
print('Theta optimizada:', theta)
```

```
<ipython-input-21-23a2650de068>:5: OptimizeWarning: Unknown solver options: maxiter
res = optimize.minimize(costFunction,
Costo con un valor de theta encontrado por optimize.minimize: 0.258
Theta optimizada: [-7.24504530e+00  1.65106640e-02  6.62394172e-01  4.42354966e-03
 4.93714565e-03  3.02202179e-01 -3.62999401e-02]
```

ihle

```
def plotDecisionBoundary(plotData, theta, X, y):
    """
    Grafica los puntos X y y en una nueva figura con un límite de decisión definido por theta.
    Grafica los puntos con * para los ejemplos positivos y con o para los ejemplos negativos.
    """
    # Hacer que theta sea un arreglo numpy
    theta = np.array(theta)

    # Graficar los datos (recordar que la primera columna en X es la intercepción)
    plotData[X[:, 1:3], y] # Ajusta para graficar las dos primeras características

    if X.shape[1] <= 3:
        # Solo se requieren 2 puntos para definir una línea, para lo cual se eligen dos puntos finales
        plot_x = np.array([np.min(X[:, 1]) - 2, np.max(X[:, 1]) + 2])

        # Calcular la línea límite de decisión
        plot_y = (-1. / theta[2]) * (theta[1] * plot_x + theta[0])

        # Graficar y ajustar los ejes para una mejor visualización
        plt.plot(plot_x, plot_y, label='Límite de decisión')
        plt.xlim([np.min(X[:, 1]) - 5, np.max(X[:, 1]) + 5])
        plt.ylim([np.min(X[:, 2]) - 5, np.max(X[:, 2]) + 5])
        plt.legend()
        plt.show()
```

Luego con eso graficamos y empezamos a hacer la evaluación de la regresión logística usando los códigos que nos enseñó en clases

z\_Alain\_Jonathan.ipynb ☆

Entorno de ejecución Herramientas Ayuda Guardado por última vez: 5:25

Comentario Compartir Gemini

```
[ ] + Código + Texto
p = np.zeros(m)
[ ] p = np.round(sigmoid(X.dot(theta.T)))
    return p
```

Una vez entrenado el modelo se procede a realizar la predicción y evaluación de los resultados de predecir cual es el valor que vota el modelo para todos los datos utilizados en el entrenamiento.

```
[ ] # Predecir la probabilidad de ingreso para un estudiante con valores específicos de las características
prob = sigmoid(np.dot([1, 45, 85, 70, 50, 60, 75], theta)) # Ajustar los valores según las características
print('Para un conjunto de características dado, se predice una probabilidad de: {:.3f}'.format(prob))

# Calcular la precisión en el conjunto de entrenamiento
p = predict(theta, X_train)
print('Precisión de entrenamiento: {:.2f} %'.format(np.mean(p == y_train) * 100))

# Calcular la precisión en el conjunto de prueba
X_test = np.concatenate([np.ones((X_test.shape[0], 1)), X_test], axis=1) # Añadir columna de unos a X_test
p_test = predict(theta, X_test)
print('Precisión de prueba: {:.2f} %'.format(np.mean(p_test == y_test) * 100))
```

↩ Para un conjunto de características dado, se predice una probabilidad de: 1.000  
Precisión de entrenamiento: 90.48 %  
Precisión de prueba: 90.70 %

Modificamos lo que es prob con datos en relación a nuestro dataset, luego logramos encontrar lo que es nuestra probabilidad que es 1 y nuestra precisión de entrenamiento y prueba.