**BỘ GIÁO DỤC & ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

**KHOA ĐIỆN – ĐIỆN TỬ**

**BỘ MÔN TỰ ĐỘNG ĐIỀU KHIỂN**

**-----------------⸙∆⸙-----------------**



**MÔN HỌC**

**TRÍ TUỆ NHÂN TẠO NÂNG CAO**

**BÀI TẬP 2**

**GVHD: Trần Vũ Hoàng**

**SVTH: Nguyễn Công Trung**

**MSSV: 2341109**

**Tp. Hồ Chí Minh tháng 07 năm 2024**

# Mục lục

[Mục lục i](#_Toc172347923)

[Danh sách hình ảnh ii](#_Toc172347924)

[Chương 1. YÊU CẦU 3](#_Toc172347925)

[Chương 2. THỰC HIỆN 4](#_Toc172347926)

[2.1. Chương trình 4](#_Toc172347927)

[2.2. Kết quả 5](#_Toc172347928)

[Chương 3. KẾT LUẬN 12](#_Toc172347929)

# Danh sách hình ảnh

[Hình 1. Learning rate = 1, số vòng lặp = 1000. 6](#_Toc172347911)

[Hình 2. Giá trị Theta tính được. 7](#_Toc172347912)

[Hình 3. Learning rate = 0.1, số vòng lặp = 1000. 8](#_Toc172347913)

[Hình 4. Giá trị Theta tính được. 9](#_Toc172347914)

[Hình 5. Learning rate = 0.01, số vòng lặp = 1000. 10](#_Toc172347915)

[Hình 6. Giá trị Theta tính được. 11](#_Toc172347916)

# YÊU CẦU

Trong tập dữ liệu tôi gửi bao gồm 118 hàng và 3 cột (phân tách nhau bằng dấu ","), trong đó:

- 118 hàng là số lượng dữ liệu (m)

- 3 cột: 2 cột đầu ( n ) là các đặc trưng (x), cột cuối

cùng là y

Các bạn làm các công việc sau:

1. Biểu diễn dữ liệu

2. Mô hình dữ liệu x theo dạng bậc 6 hai biến như

hình bên cạnh

3. Chuẩn hóa dữ liệu

4. Viết chương trình cho phép học các tham số của

mô hình phân loại phi tuyến trên

5. Tính J ở mỗi vòng lặp, và vẽ biểu đồ J ở các giá

trị learning rate khác nhau sau khi chạy hết các

vòng lặp.

# THỰC HIỆN

## Chương trình

import polars as pl

import numpy as np

import matplotlib.pyplot as plt

# Open and read the text file

with open('ex5data.txt', 'r') as file:

lines = file.readlines()

# Split each line by commas and create a list of lists

split\_data = [line.strip().split(',') for line in lines]

# Convert the split data into a list of tuples with numeric values

numeric\_data = [(float(row[0]), float(row[1]), float(row[2])) for row in split\_data]

# Specify column names

column\_names = ['x1', 'x2', 'y']

# Create a Polars DataFrame with specified schema

data = pl.DataFrame(numeric\_data, schema=[(col, pl.Float64) for col in column\_names], orient="row")

columns = data.columns

n = len(columns)-1

m = len(data)

x = data.select(columns[:n])

y = data.select(columns[-1])

x1 = data['x1'].to\_list()

x2 = data['x2'].to\_list()

y\_list = data['y'].to\_list()

plt.figure()

plt.scatter([x1[i] for i in range(len(y\_list)) if y\_list[i] == 0], [x2[i] for i in range(len(y\_list)) if y\_list[i] == 0], color='blue', label='Class 0')

plt.scatter([x1[i] for i in range(len(y\_list)) if y\_list[i] == 1], [x2[i] for i in range(len(y\_list)) if y\_list[i] == 1], color='red', label='Class 1')

plt.xlabel('x1')

plt.ylabel('x2')

plt.legend()

plt.title('Scatter plot of x1 vs x2 colored by y')

degrees = 6

results = []

for d in range(1, degrees+1):

for i in range(d, -1, -1):

term = x['x1'] \*\* i \* x['x2'] \*\* (d - i)

results.append(term)

# Exclude the first two columns from the results

results = results[2:]

# Convert the accumulated results to Polars Series

for idx in range(len(results)):

x = x.with\_columns(pl.Series(f'res{idx}', results[idx]))

# Display the updated DataFrame

print(x)

columns = x.columns

n = len(columns)

mean\_x = x.mean()

std\_x = x.std()

standardized\_x = x.with\_columns([(pl.col(col) - mean\_x[col]) / std\_x[col] for col in columns[:n]])

standardized\_x = standardized\_x.with\_columns(pl.lit(1).alias("constant"))

standardized\_x = standardized\_x.select(["constant"] + [col for col in standardized\_x.columns if col != "constant"])

n

x\_mat = standardized\_x.to\_numpy().reshape(-1, n+1).T

y\_mat = y.to\_numpy().reshape(-1, 1)

theta = np.random.rand(n+1, 1)

H = 1 / (1 + np.exp(-np.dot(theta.T, x\_mat))).T

alpha = 1

loop = 1000

J\_values = np.zeros((loop, 1))

for i in range(loop):

theta = theta - (alpha/m)\*np.dot(x\_mat, (H-y\_mat))

H = 1 / (1 + np.exp(-np.dot(theta.T, x\_mat))).T

J\_values[i] = 1/(m)\*(-np.dot(y\_mat.T, np.log(H))-np.dot(1-y\_mat.T, np.log(1-H)))

standardized\_data = standardized\_x.with\_columns(y)

print("Theta Values: ")

print(theta)

# Create a figure and a set of subplots

fig, axs = plt.subplots()

# Plot the cost function on the first subplot

axs.plot(J\_values)

axs.set\_xlabel('Iteration')

axs.set\_ylabel('Cost (J)')

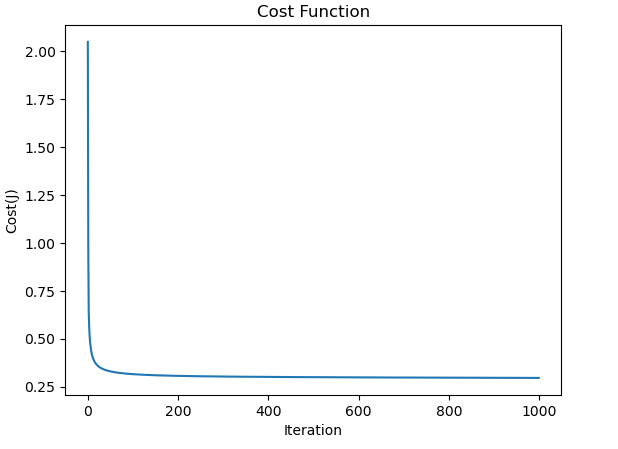
axs.set\_title('Cost Function')

# Layout so plots do not overlap

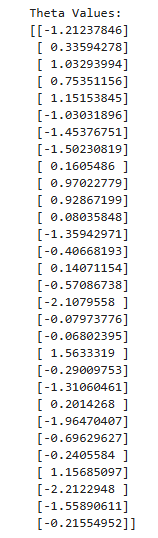
fig.tight\_layout()

## Kết quả

* Thông số learning rate = 1, số vòng lặp = 1000.

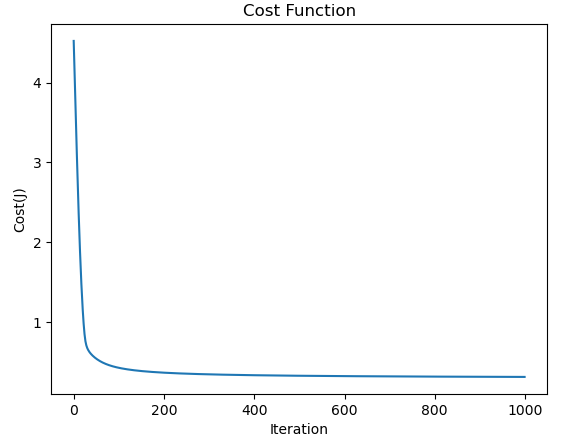


Hình . Learning rate = 1, số vòng lặp = 1000.

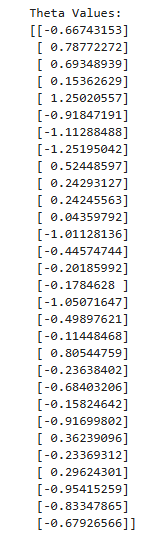


Hình . Giá trị Theta tính được.

* Thông số learning rate = 0.1, số vòng lặp = 1000.

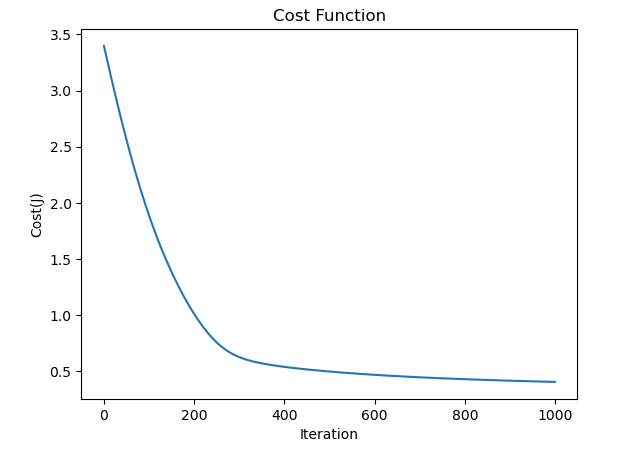


Hình . Learning rate = 0.1, số vòng lặp = 1000.

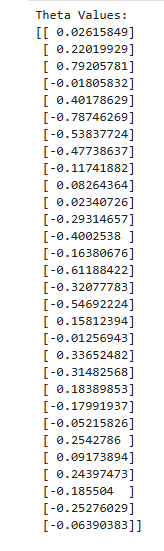


Hình . Giá trị Theta tính được.

* Thông số learning rate = 0.01, số vòng lặp = 1000.



Hình . Learning rate = 0.01, số vòng lặp = 1000.



Hình . Giá trị Theta tính được.

# KẾT LUẬN

Chuẩn hóa dữ liệu là một bước quan trọng trong quá trình huấn luyện mô hình học máy, giúp các đặc tính của dữ liệu có cùng thang đo, thường là mean bằng 0 và variance bằng 1. Điều này làm cho quá trình tối ưu hóa bằng phương pháp gradient descent hiệu quả hơn, cho phép tăng tốc độ học mà không lo vượt qua giá trị tối ưu, dẫn đến hội tụ nhanh hơn. Gradient descent là phương pháp tối ưu hóa theo từng bước nhỏ dựa trên gradient của hàm mất mát để tìm ra điểm tối ưu. Khi dữ liệu được chuẩn hóa và tốc độ học được điều chỉnh phù hợp, quá trình hội tụ trở nên nhanh chóng và hiệu quả hơn.