**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**

**GVHD: Trần Vũ Hoàng**

**SVTH: Nguyễn Công Trung**

**MSSV: 2341109**

**Tp. Hồ Chí Minh tháng 07 năm 2024**

# Mục lục

[Mục lục i](#_Toc173508210)

[Danh sách hình ảnh ii](#_Toc173508211)

[Chương 1. YÊU CẦU 3](#_Toc173508212)

[Chương 2. THỰC HIỆN 4](#_Toc173508213)

[2.1. Chương trình 4](#_Toc173508214)

[2.2. Kết quả 7](#_Toc173508215)

# Danh sách hình ảnh

[Hình 1. Lamda = 1 8](#_Toc173508226)

[Hình 2. Lamda = 0.1 8](#_Toc173508227)

[Hình 3. Lamda = 0.01 9](#_Toc173508228)

[Hình 4. Lamda = 1 9](#_Toc173508229)

[Hình 5. Lamda = 0.1 10](#_Toc173508230)

[Hình 6. Lamda = 0.01 10](#_Toc173508231)

# YÊU CẦU

Các bạn sử dụng tập dữ liệu của bài tập 5, mô hình dữ liệu x theo dạng bậc 6

và làm các công việc sau:

1. Chia dữ liệu ra thành 2 tập: training (70%) và validation (30%). Phải đảm bảo việc chia dữ liệu là ngẫu nhiên và tỷ lệ positive và negative cân bằng.

2. Viết chương trình cho phép học các tham số của mô hình phân loại phi

tuyến trên có sử dụng regularization L2 và L1.

3. Tính J ở mỗi vòng lặp cho cả hai tập, chọn điểm dừng phù hợp.

4. Thay đổi lambda và tính J cho mỗi lambda tương ứng cho cả hai tập. Vẽ biểu đồ quan hệ giữa J và lambda từ đó lựa chọn lambda phù hợp.

# THỰC HIỆN

## Chương trình

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  # Load the data  df = pd.read\_csv("ex5data.txt", header=None, sep=",")  x = df.values[:, :-1]  y = df.values[:, -1]  m = len(y)  # Separate positive and negative examples  x\_pos = x[y == 1, :]  x\_neg = x[y == 0, :]  # Calculate the number of examples for training (70%) and testing (30%)  num\_pos\_train = int(0.7 \* len(x\_pos))  num\_neg\_train = int(0.7 \* len(x\_neg))  num\_pos\_test = len(x\_pos) - num\_pos\_train  num\_neg\_test = len(x\_neg) - num\_neg\_train  # Set the random seed for reproducibility  np.random.seed(0)  # Shuffle the indices  pos\_indices = np.random.permutation(len(x\_pos))  neg\_indices = np.random.permutation(len(x\_neg))  # Select training and testing indices  train\_pos\_indices = pos\_indices[:num\_pos\_train]  train\_neg\_indices = neg\_indices[:num\_neg\_train]  test\_pos\_indices = pos\_indices[num\_pos\_train:]  test\_neg\_indices = neg\_indices[num\_neg\_train:]  # Create the DataFrames  train\_X\_ori = np.vstack((x\_pos[train\_pos\_indices], x\_neg[train\_neg\_indices]))  train\_y = np.hstack((np.ones(num\_pos\_train), np.zeros(num\_neg\_train)))  test\_X\_ori = np.vstack((x\_pos[test\_pos\_indices], x\_neg[test\_neg\_indices]))  test\_y = np.hstack((np.ones(num\_pos\_test), np.zeros(num\_neg\_test)))  # Plotting the data  plt.scatter(x\_pos[:, 0], x\_pos[:, 1], c='r', label='Positive')  plt.scatter(x\_neg[:, 0], x\_neg[:, 1], c='k', label='Negative')  plt.xlabel('Feature 1')  plt.ylabel('Feature 2')  plt.legend()  plt.show() |

|  |
| --- |
| def mapfeature(x1, x2, m, degree):      X = np.ones((m, 1))      for i in range(1, degree+1):          for j in range(i+1):              X = np.concatenate((X, np.reshape((x1\*\*(i-j))\*x2\*\*j, (m,1))), 1)      return X  degree = 6  train\_X = mapfeature(train\_X\_ori[:,0], train\_X\_ori[:,1], train\_X\_ori.shape[0], degree)  test\_X = mapfeature(test\_X\_ori[:,0], test\_X\_ori[:,1], test\_X\_ori.shape[0], degree) |

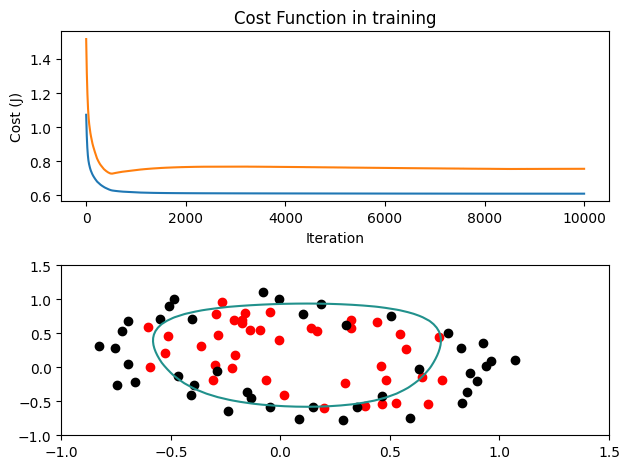
|  |
| --- |
| # Regularized logistic regression L2  ite = 10000  lr = 0.1  lambda\_ = 1  # Regularization parameter  J\_train = np.zeros(ite)  J\_test = np.zeros(ite)  train\_y = np.reshape(train\_y, (len(train\_y), 1))  test\_y = np.reshape(test\_y, (len(test\_y), 1))  np.random.seed(2)  theta = np.random.rand(train\_X.shape[1], 1)  m\_train = len(train\_y)  m\_test = len(test\_y)  z = np.dot(train\_X, theta)  h = 1.0 / (1.0 + np.exp(-z))  z\_test = np.dot(test\_X, theta)  h\_test = 1.0 / (1.0 + np.exp(-z\_test))  for i in range(ite):      theta\_tmp = theta.copy()      theta\_tmp[0] = 0      gradient = (1 / m\_train) \* np.dot(train\_X.T, (h - train\_y)) + (lambda\_ / m\_train) \* theta\_tmp      theta = theta - lr \* gradient      theta\_tmp = theta.copy()      theta\_tmp[0] = 0      z = np.dot(train\_X, theta)      h = 1.0 / (1.0 + np.exp(-z))      z\_test = np.dot(test\_X, theta)      h\_test = 1.0 / (1.0 + np.exp(-z\_test))      J\_train[i] = np.mean(-train\_y\*np.log(h)-(1-train\_y)\*np.log(1-h)) + (lambda\_ / (2 \* m\_train)) \* np.sum(np.square(theta\_tmp))      J\_test[i] = np.mean(-test\_y\*np.log(h\_test)-(1-test\_y)\*np.log(1-h\_test)) + (lambda\_ / (2 \* m\_test)) \* np.sum(np.square(theta\_tmp))  # Create a figure and a set of subplots  fig, axs = plt.subplots()  axs.plot(J\_train)  axs.plot(J\_test)  axs.set\_xlabel('Iteration')  axs.set\_ylabel('Cost (J)')  axs.set\_title('Cost Function in training')  # Layout so plots do not overlap  fig.tight\_layout()  plt.scatter(x\_pos[train\_pos\_indices][:,0], x\_pos[train\_pos\_indices][:,1], c='r')  plt.scatter(x\_neg[train\_neg\_indices][:,0], x\_neg[train\_neg\_indices][:,1],c='k')  u = np.linspace(-1, 1.5, 50)  v = np.linspace(-1, 1.5, 50)  z = np.zeros((len(u),len(v)))  for i in range(len(u)):      for j in range(len(v)):          z[i, j] = np.dot(mapfeature(u[i], v[j], 1, degree), theta)  z = z.T  plt.contour(u, v, z, 0) |

|  |
| --- |
| # Regularized logistic regression L1  ite = 500  lr = 0.1  lambda\_ = 1  J\_train = np.zeros(ite)  J\_test = np.zeros(ite)  train\_y = np.reshape(train\_y, (len(train\_y), 1))  test\_y = np.reshape(test\_y, (len(test\_y), 1))  np.random.seed(2)  theta = np.random.rand(train\_X.shape[1], 1)  m\_train = len(train\_y)  m\_test = len(test\_y)  z = np.dot(train\_X, theta)  h = 1.0 / (1.0 + np.exp(-z))  z\_test = np.dot(test\_X, theta)  h\_test = 1.0 / (1.0 + np.exp(-z\_test))  for i in range(ite):      theta\_tmp = theta.copy()      theta\_tmp[0] = 0      gradient = (1 / m\_train) \* np.dot(train\_X.T, (h - train\_y)) + (lambda\_ / m\_train) \* np.sign(theta\_tmp)      theta = theta - lr \* gradient      theta\_tmp = theta.copy()      theta\_tmp[0] = 0      z = np.dot(train\_X, theta)      h = 1.0 / (1.0 + np.exp(-z))      z\_test = np.dot(test\_X, theta)      h\_test = 1.0 / (1.0 + np.exp(-z\_test))      J\_train[i] = np.mean(-train\_y\*np.log(h)-(1-train\_y)\*np.log(1-h)) + (lambda\_ / m\_train) \* np.sum(np.abs(theta\_tmp))      J\_test[i] = np.mean(-test\_y\*np.log(h\_test)-(1-test\_y)\*np.log(1-h\_test)) + (lambda\_ / m\_test) \* np.sum(np.abs(theta\_tmp))  # Create a figure and a set of subplots  fig, axs = plt.subplots()  axs.plot(J\_train)  axs.plot(J\_test)  axs.set\_xlabel('Iteration')  axs.set\_ylabel('Cost (J)')  axs.set\_title('Cost Function in training')  # Layout so plots do not overlap  fig.tight\_layout()  plt.scatter(x\_pos[train\_pos\_indices][:,0], x\_pos[train\_pos\_indices][:,1], c='r')  plt.scatter(x\_neg[train\_neg\_indices][:,0], x\_neg[train\_neg\_indices][:,1],c='k')  u = np.linspace(-1, 1.5, 50)  v = np.linspace(-1, 1.5, 50)  z = np.zeros((len(u),len(v)))  for i in range(len(u)):      for j in range(len(v)):          z[i, j] = np.dot(mapfeature(u[i], v[j], 1, degree), theta)  z = z.T  plt.contour(u, v, z, 0) |

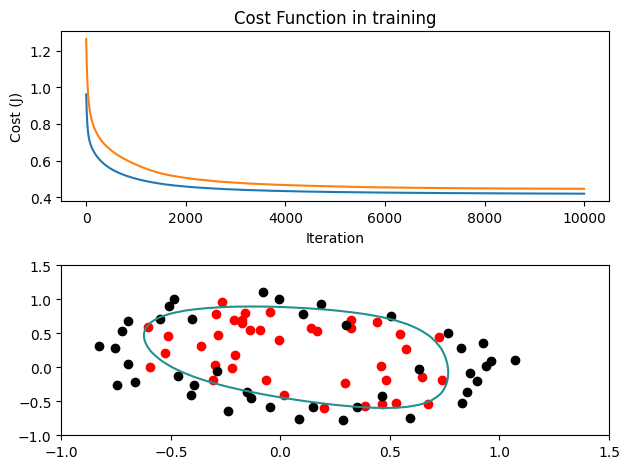
## Kết quả

Thông số learning rate = 0.1, số vòng lặp = 10000.

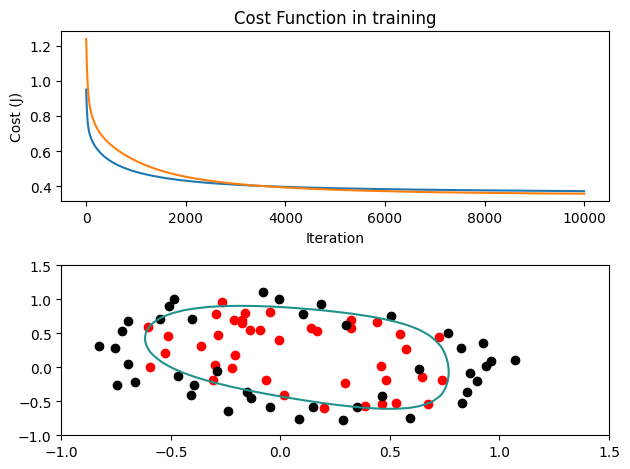
2.2.1. Trường hợp L1:



Hình . Lamda = 1

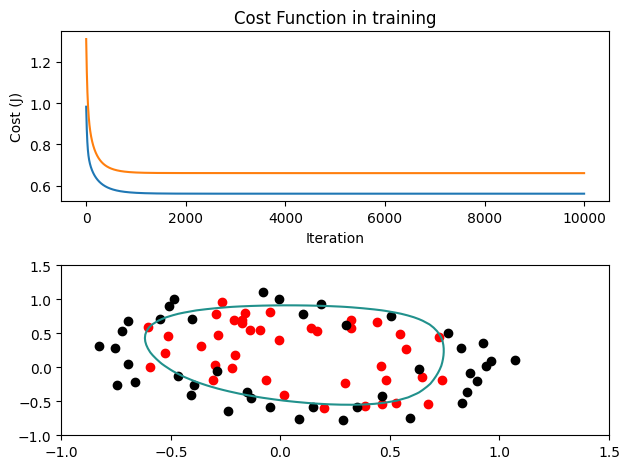


Hình . Lamda = 0.1

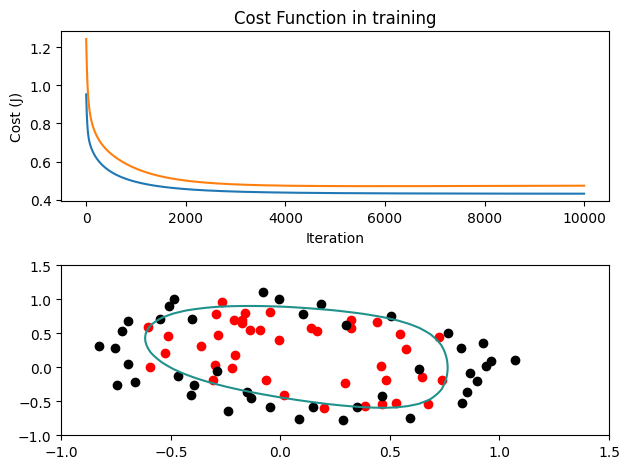


Hình . Lamda = 0.01

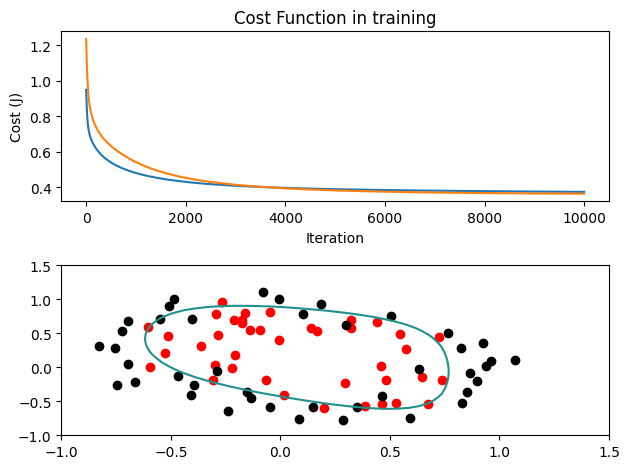
2.2.2. Trường hợp L2:



Hình . Lamda = 1



Hình . Lamda = 0.1



Hình . Lamda = 0.01