Report of Assignment_1_part_2

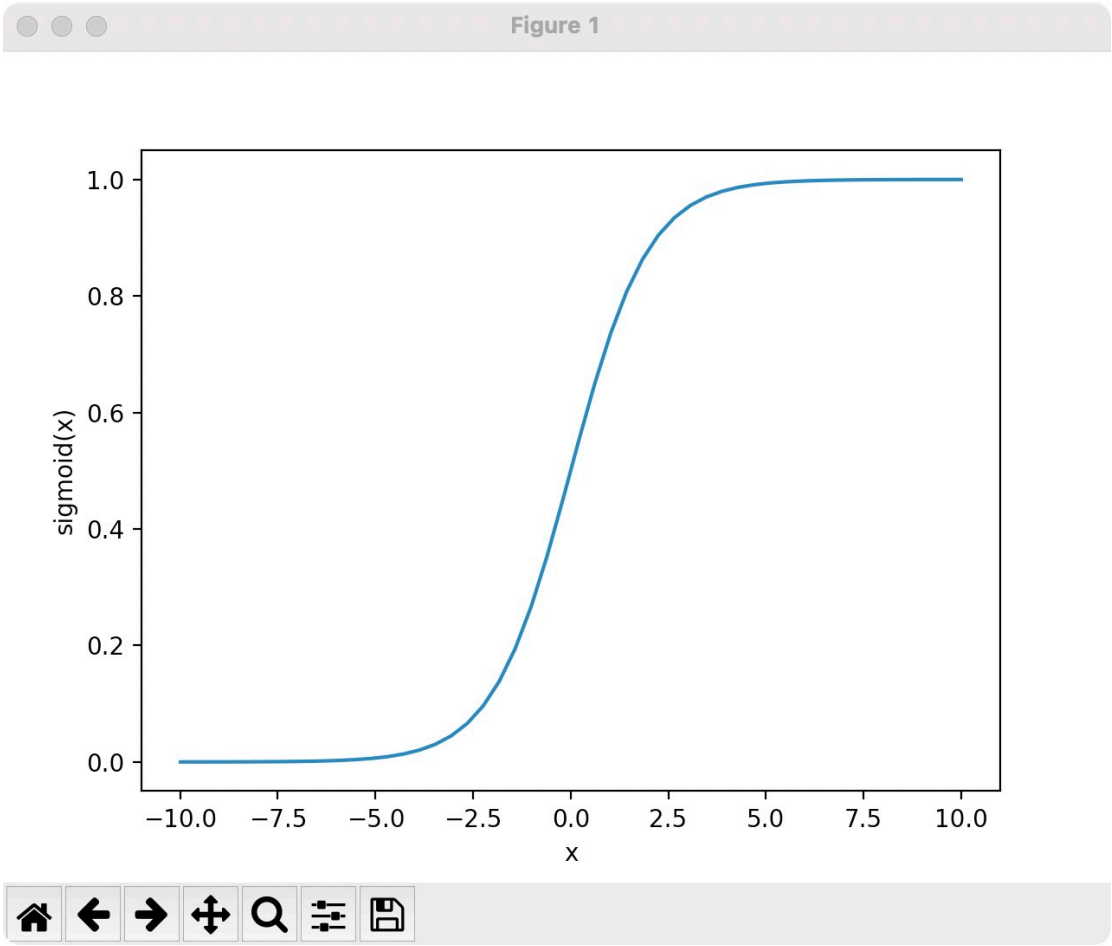
1. logistic Regression

   Task 1 :

   Fill out the sigmoid.py function and use the plot_sigmoid.py to plot:

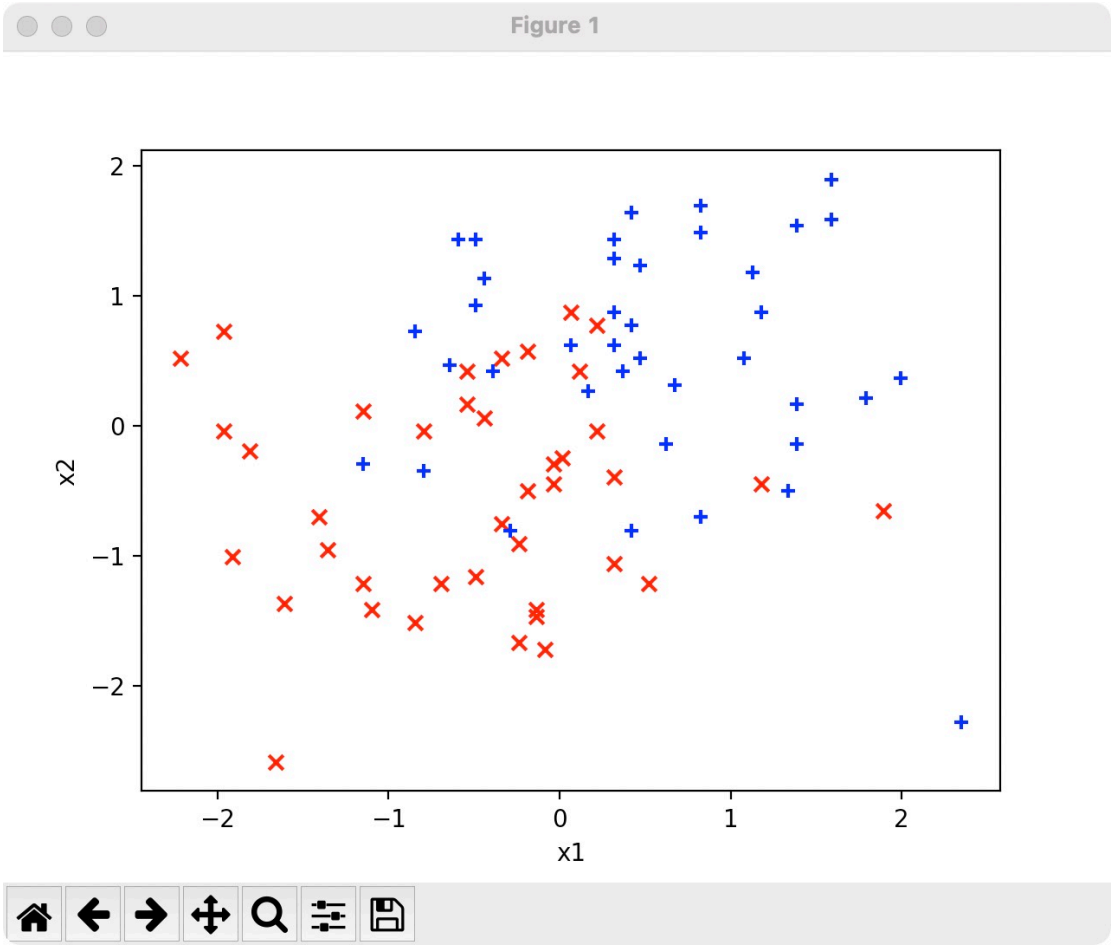
```python
def sigmoid(z):

    output = 0.0
    ########################################
    # Write your code here
    # modify this to return z passed through the sigmoid function
    output=1/(1 + np.exp(-z))
    ########################################/

    return output
```

```python
import numpy as np
import matplotlib.pyplot as plt
from sigmoid import *

def plot_sigmoid():
    x = np.linspace(1,2000)/100.0 - 10
    y = sigmoid(x)
    fig, ax1 = plt.subplots()
    ax1.plot(x, y)
    # set label of horizontal axis
    ax1.set_xlabel('x')
    # set label of vertical axis
    ax1.set_ylabel('sigmoid(x)')
    plt.show()

plot_sigmoid()
```

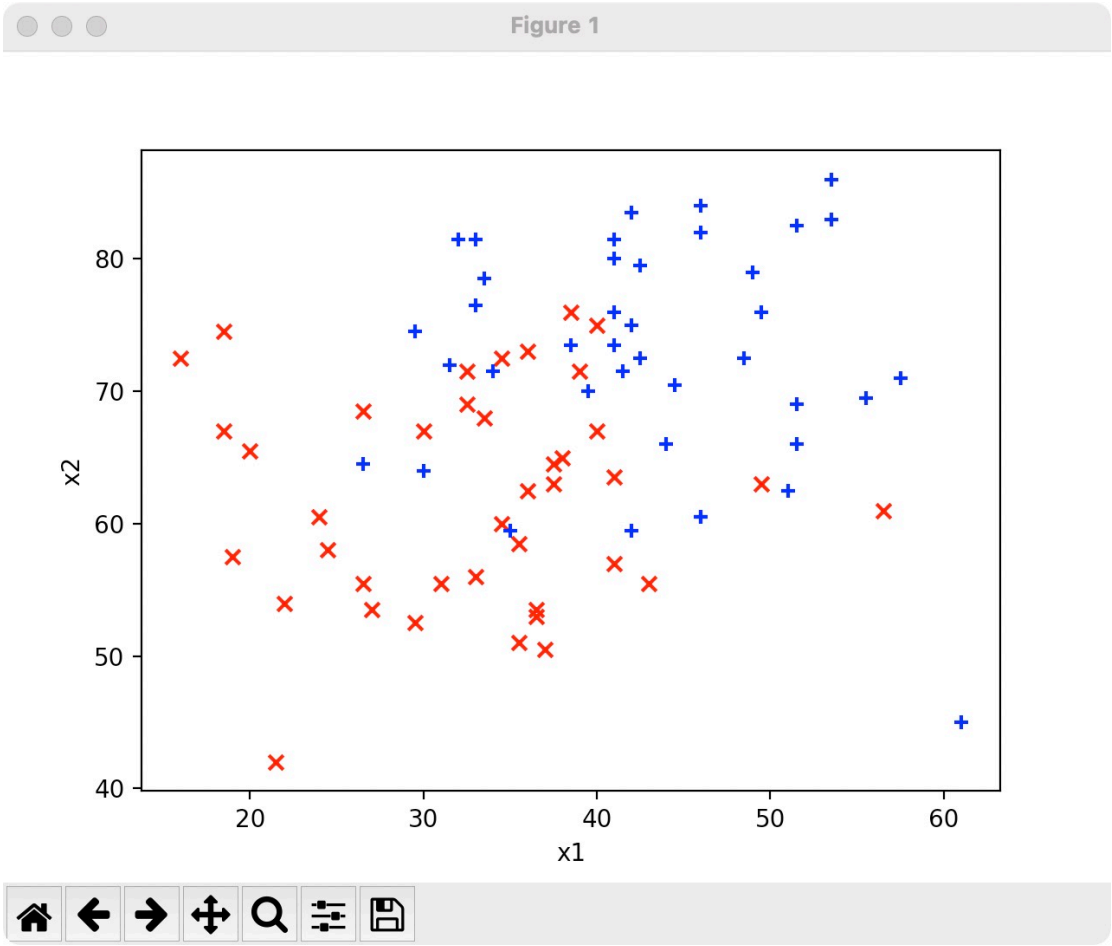


Task 2:

Plot the normalized data:



Plot the data without normalized:



Task 3:

Modify the calculate_hypothesis.py :

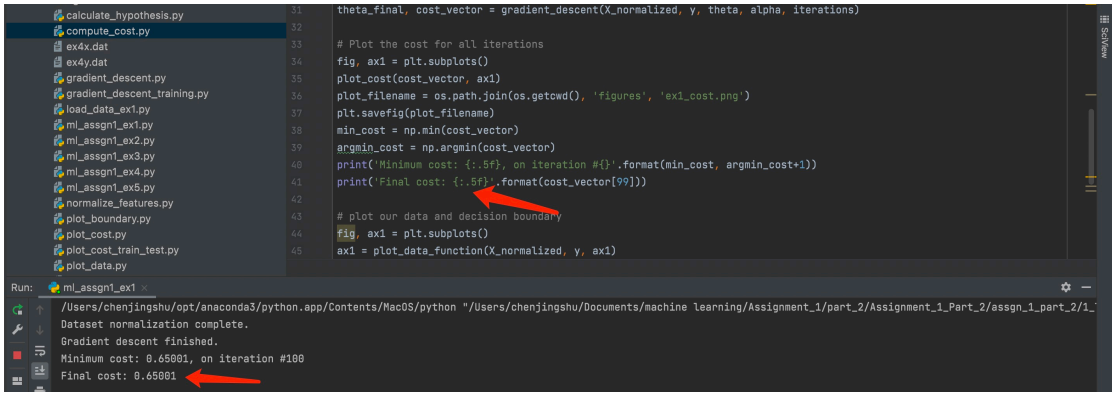
```
def calculate_hypothesis(X, theta, i):
    """
        :param X            : 2D array of our dataset
        :param theta        : 1D array of the trainable parameters
        :param i            : scalar, index of current training sample's row
    """
    hypothesis = 0.0
    ##########################################
    # Write your code here
    # You must calculate the hypothesis for the i-th sample of X, given X, theta and i.
    hypothesis = X[i,:] * np.transpose(theta)
    hypothesis = sum(hypothesis)


    ##########################################/
```

Task 4:

Modify the “cost = 0.0” in compute_cost.py:

```
# Compute cost for logistic regression.
for i in range(m):
    hypothesis = calculate_hypothesis(X, theta, i)
    output = y[i]
    #cost = 0.0
    ########################################
    # Write your code here
    # You must calculate the cost
    cost = -output * np.log(hypothesis) - (1-output) * np.log(1-hypothesis)
    ########################################/
    J += cost
J = J/m
```

The final cost found by the gradient descent:

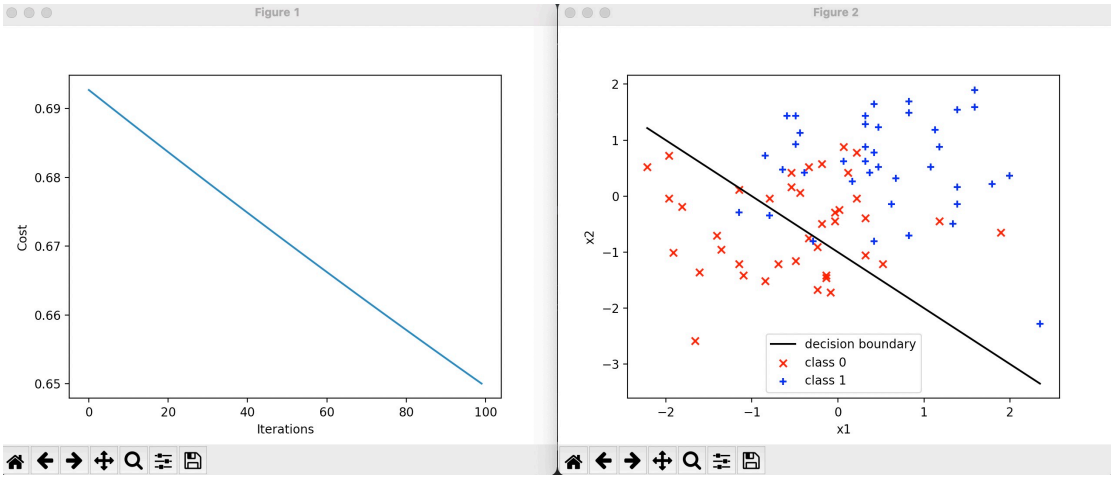


Task 5 :

Modify the plot_boundary function:

```
    ########################################
    # Write your code here
    # Re-arrange the terms in the equation of the hypothesis function, and solve with respect to x2, to find its values on
    min_x1=np.min(X[:,1])
    max_x1=np.max(X[:,1])
    x2_on_min_x1 = -(theta[0]*1+theta[1]*min_x1)/theta[2]
    x2_on_max_x1 = -(theta[0]*1+theta[1]*max_x1)/theta[2]
    ########################################/

    x_array = np.array([min_x1, max_x1])
    y_array = np.array([x2_on_min_x1, x2_on_max_x1])
    ax1.plot(x_array, y_array, c='black', label='decision boundary')

    # add legend to the subplot
    ax1.legend()
```

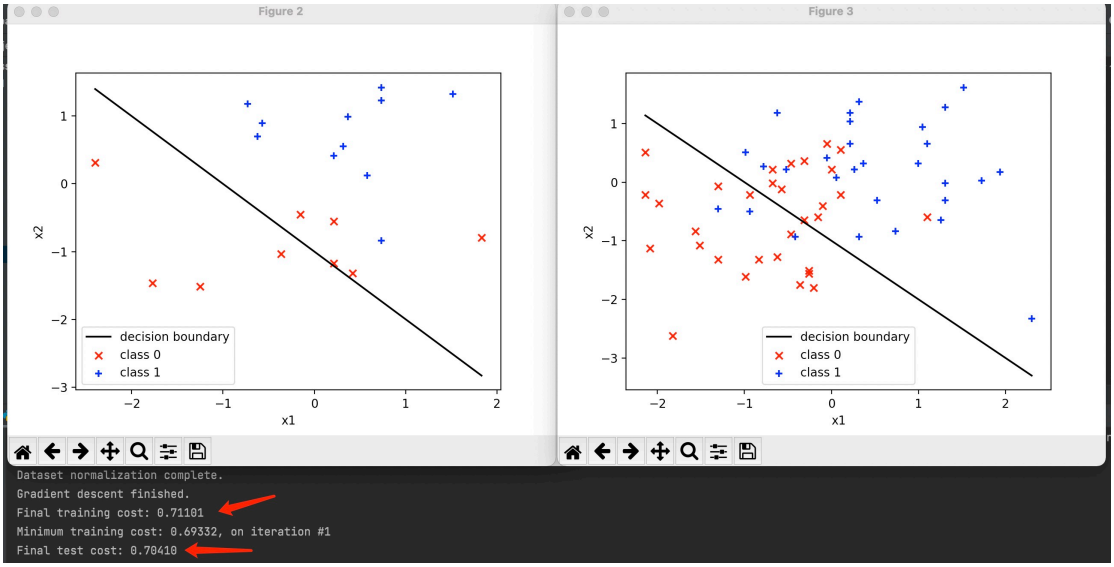
Run the assgn1_ex1.py :



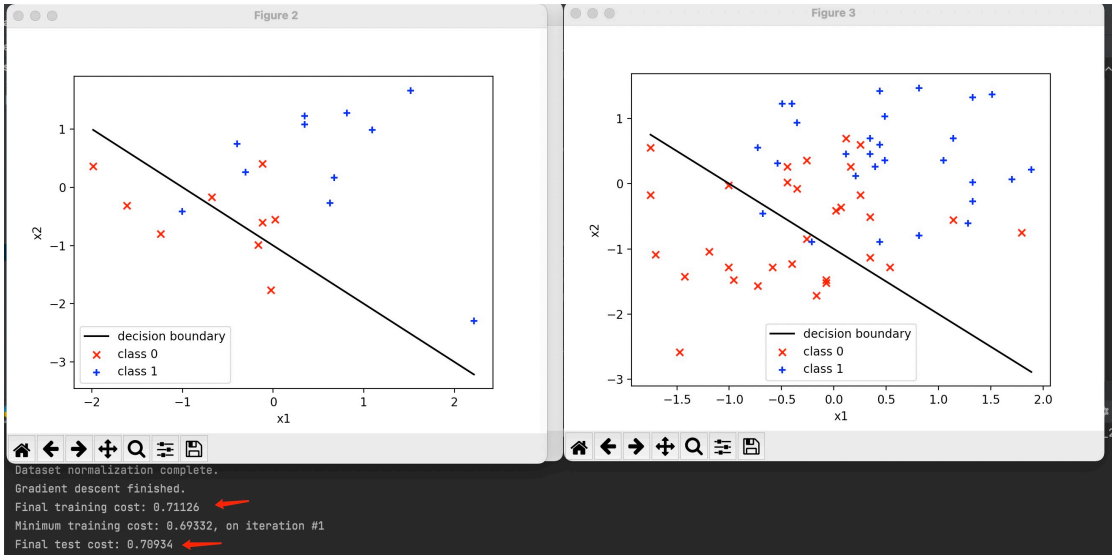
Task 6:

Run the assgn1_ex2.py several times:

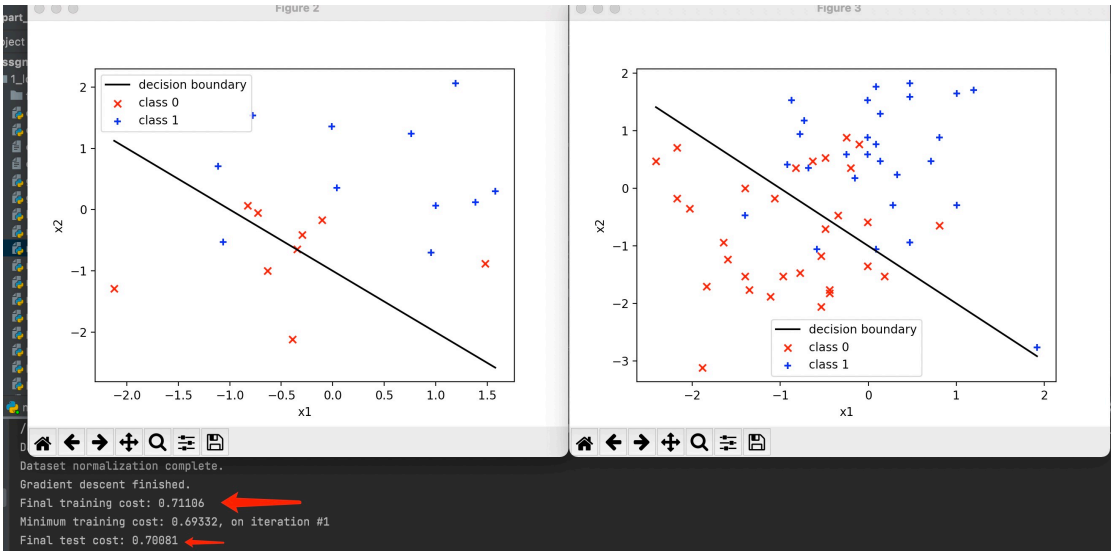
(1)



(2)



(3)



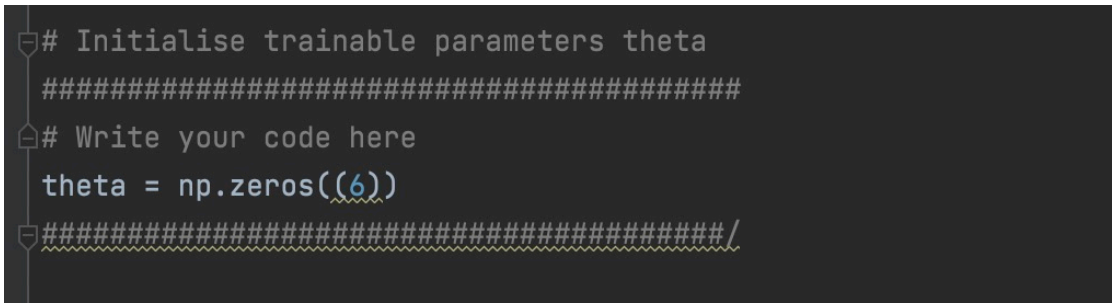
The general difference between the training and test cost is the value of training cost are all higher than the value of test cost.

When the training set can present the overview the dataset, it will generalize well.

Modify the assgn1_ex3.py to incorporate the new non-linear features:

```
#写一个for循环,新建一些列表
columns_x1x2 = np.array([], dtype=np.float32)
columns_doublex1 = np.array([], dtype=np.float32)
columns_doublex2 =np.array([], dtype=np.float32)
for i in range(X.shape[0]):
    curren_Value = X[i,0]*X[i,1]
    columns_x1x2 = np.append(columns_x1x2,curren_Value)

    curren_Value = X[i,0]*X[i,0]
    columns_doublex1 = np.append(columns_doublex1,curren_Value)

    curren_Value = X[i,1]*X[i,1]
    columns_doublex2 = np.append(columns_doublex2,curren_Value)
X=np.c_[X,columns_x1x2]
X=np.c_[X,columns_doublex1]
X=np.c_[X,columns_doublex2]
#########################################/
```

for i in range(X.shape[0])

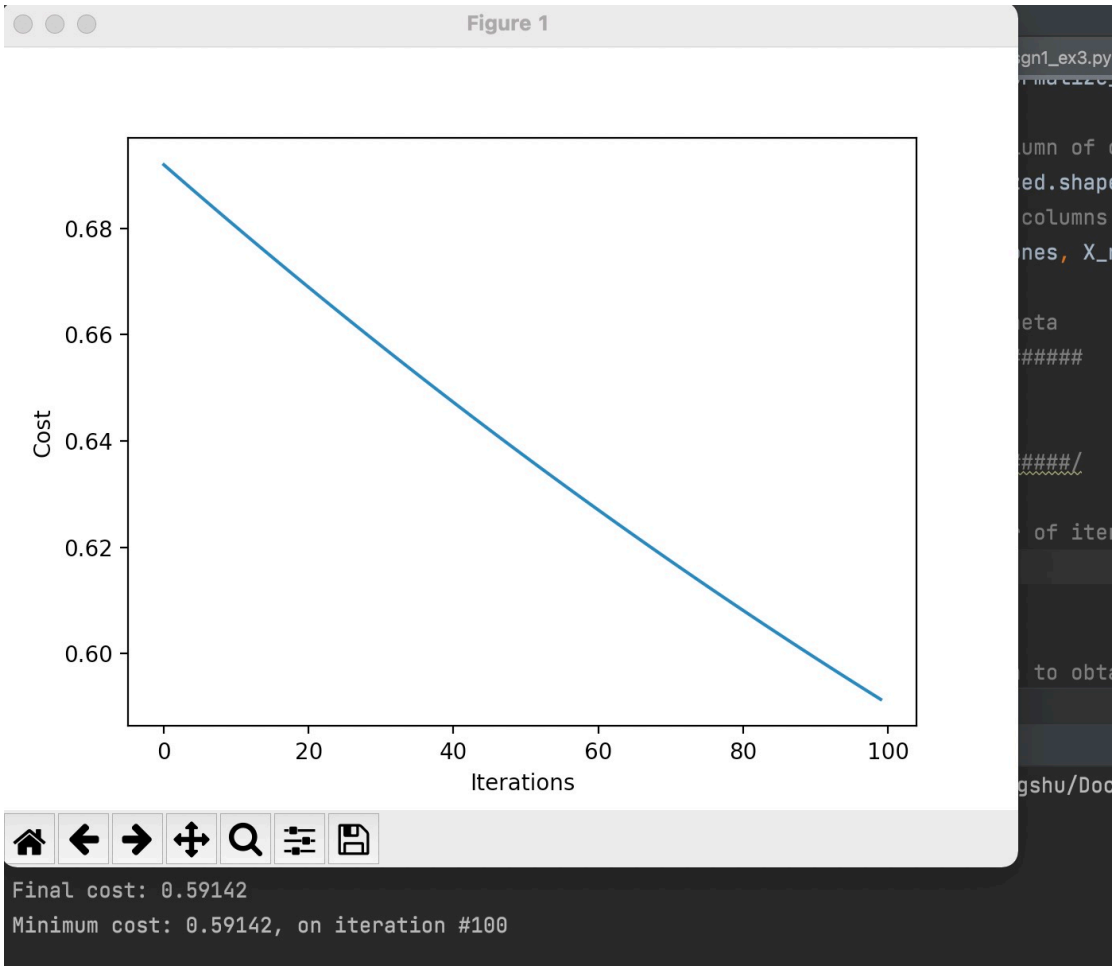


```
# Initialise trainable parameters theta
##########################################
# Write your code here
theta = np.zeros((6))
##########################################/
```

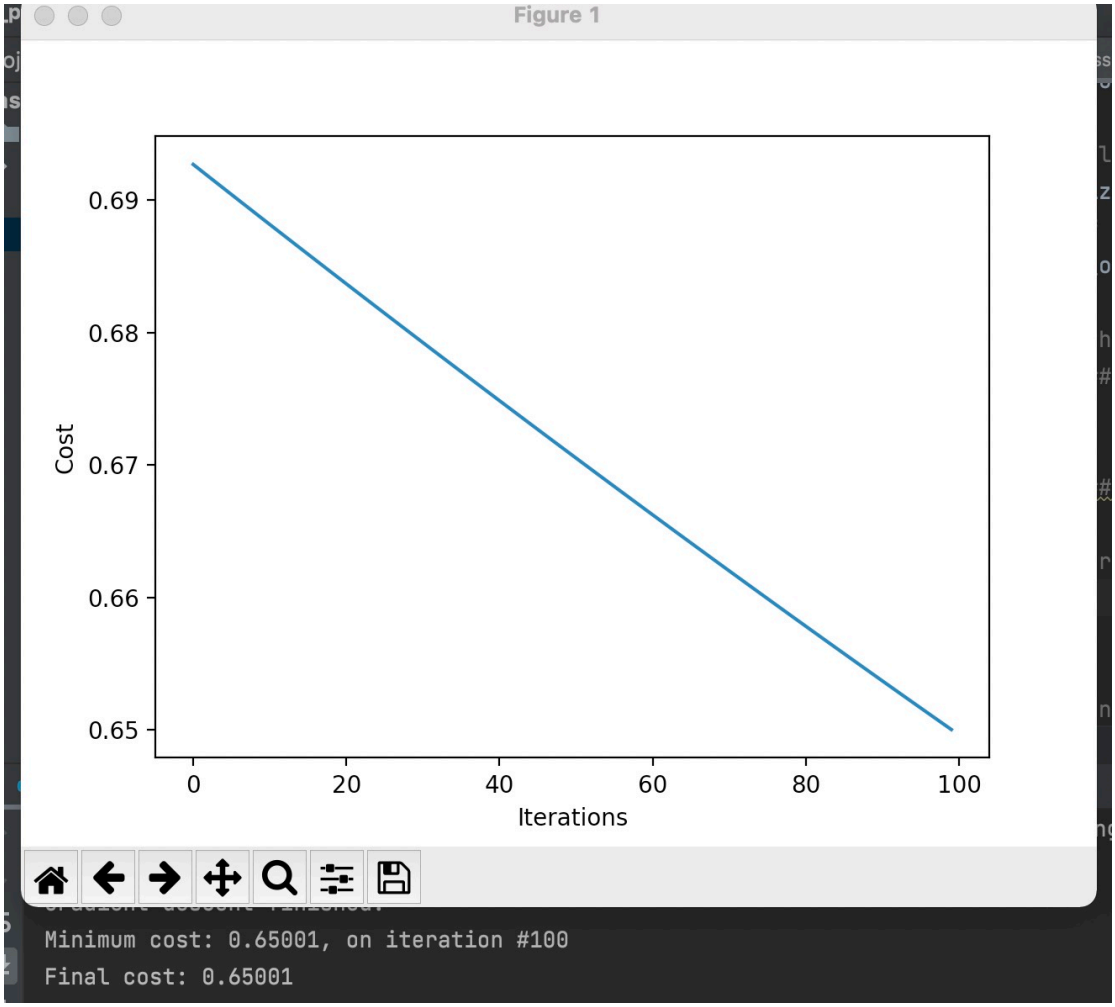
Task 7:

The error value which used 5 features is higher than the error value which used original features.

5 features:



Original features:



The value of error is lower than the value of error which used original features.

Task 8:

Modify assgn1_ex4.py to add new features and modify the gradient_descent_trainning.py to store current cost:

```python
# Append columns of the new features to the dataset, to the dimension of columns (i.e., 1)
columns_x1x2 = np.array([], dtype=np.float32)
columns_doublex1 = np.array([], dtype=np.float32)
columns_doublex2 =np.array([], dtype=np.float32)
for i in range(X.shape[0]):
    curren_Value = X[i,0]*X[i,1]
    columns_x1x2 = np.append(columns_x1x2,curren_Value)

    curren_Value = X[i,0]*X[i,0]
    columns_doublex1 = np.append(columns_doublex1,curren_Value)

    curren_Value = X[i,1]*X[i,1]
    columns_doublex2 = np.append(columns_doublex2,curren_Value)
X=np.c_[X,columns_x1x2]
X=np.c_[X,columns_doublex1]
X=np.c_[X,columns_doublex2]
```

```python
            # Calculate the hypothesis for the i-th sample of X, with a call to the "calculate_hypothesis" function
            hypothesis = calculate_hypothesis(X_train, theta, i)
            ########################################/
            output = y_train[i]
            ########################################
            # Write your code here
            # Adapt the code, to compute the values of sigma for all the elements of theta
            sigma = -(output * np.log(hypothesis) -(1-output) * np.log(1-hypothesis))
            ########################################/

        # update theta_temp
        ########################################
        # Write your code here
        # Update theta_temp, using the values of sigma
        theta_temp = theta_temp - (alpha / m) * sigma
        ########################################/
```

```python
    # append current iteration's cost to cost vector
    ########################################
    # Write your code here
    # Store costs for both train and test set in their corresponding vectors
    iteration_cost = compute_cost(X_train, y_train, theta)
    cost_vector_train = np.append(cost_vector_train, iteration_cost)

    iteration_cost = compute_cost(X_test, y_test, theta)
    cost_vector_test = np.append(cost_vector_test, iteration_cost)
    ########################################/
```

Experiment with different sizes:

(1) 60-20

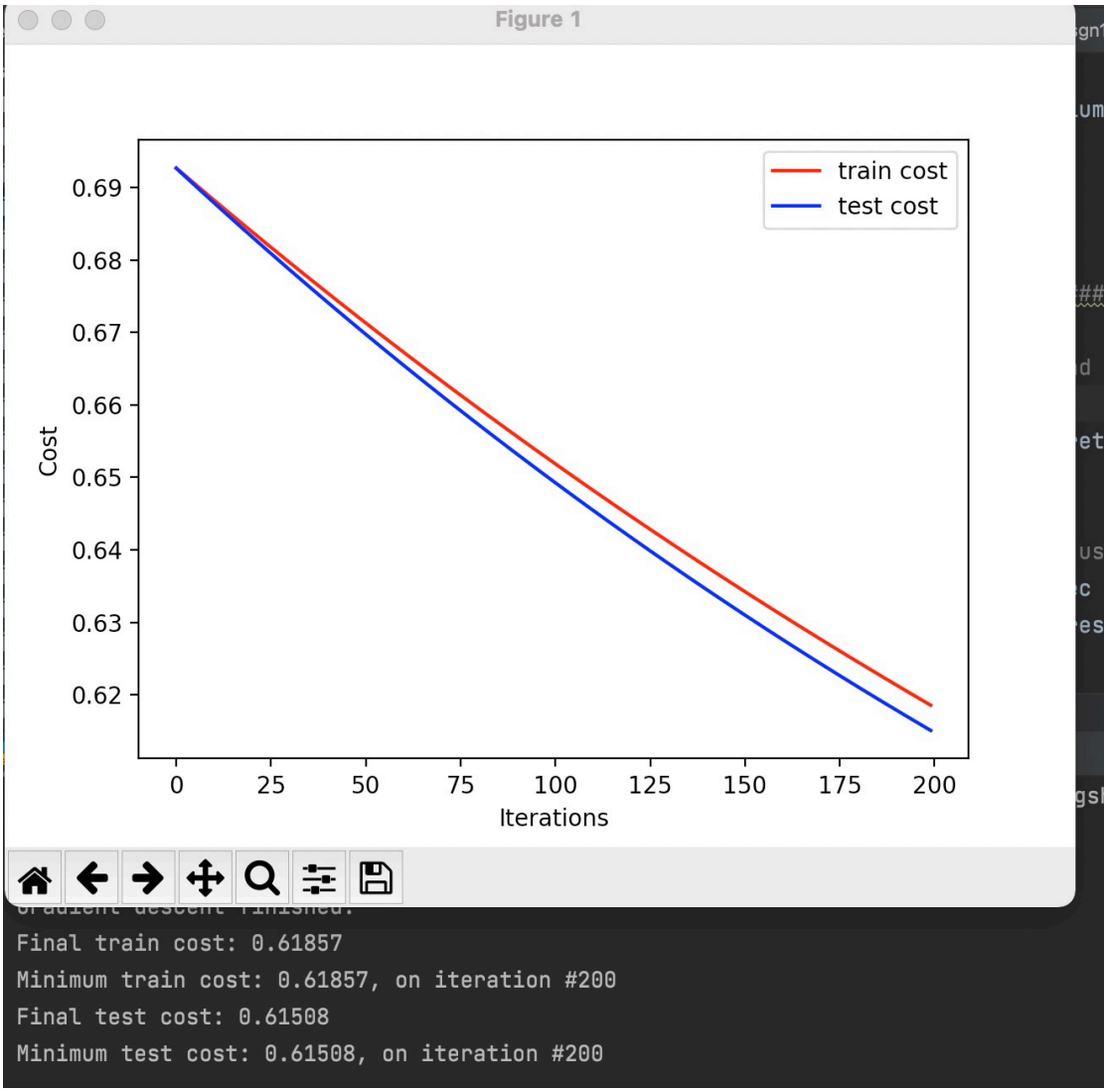


Figure 1
train cost
test cost
0.69
0.68
0.67
0.66
0.65
0.64
0.63
0.62
Cost
0
25
50
75
100
125
150
175
200
Iterations
Final train cost: 0.61857
Minimum train cost: 0.61857, on iteration #200
Final test cost: 0.61508
Minimum test cost: 0.61508, on iteration #200

(2) 70-10

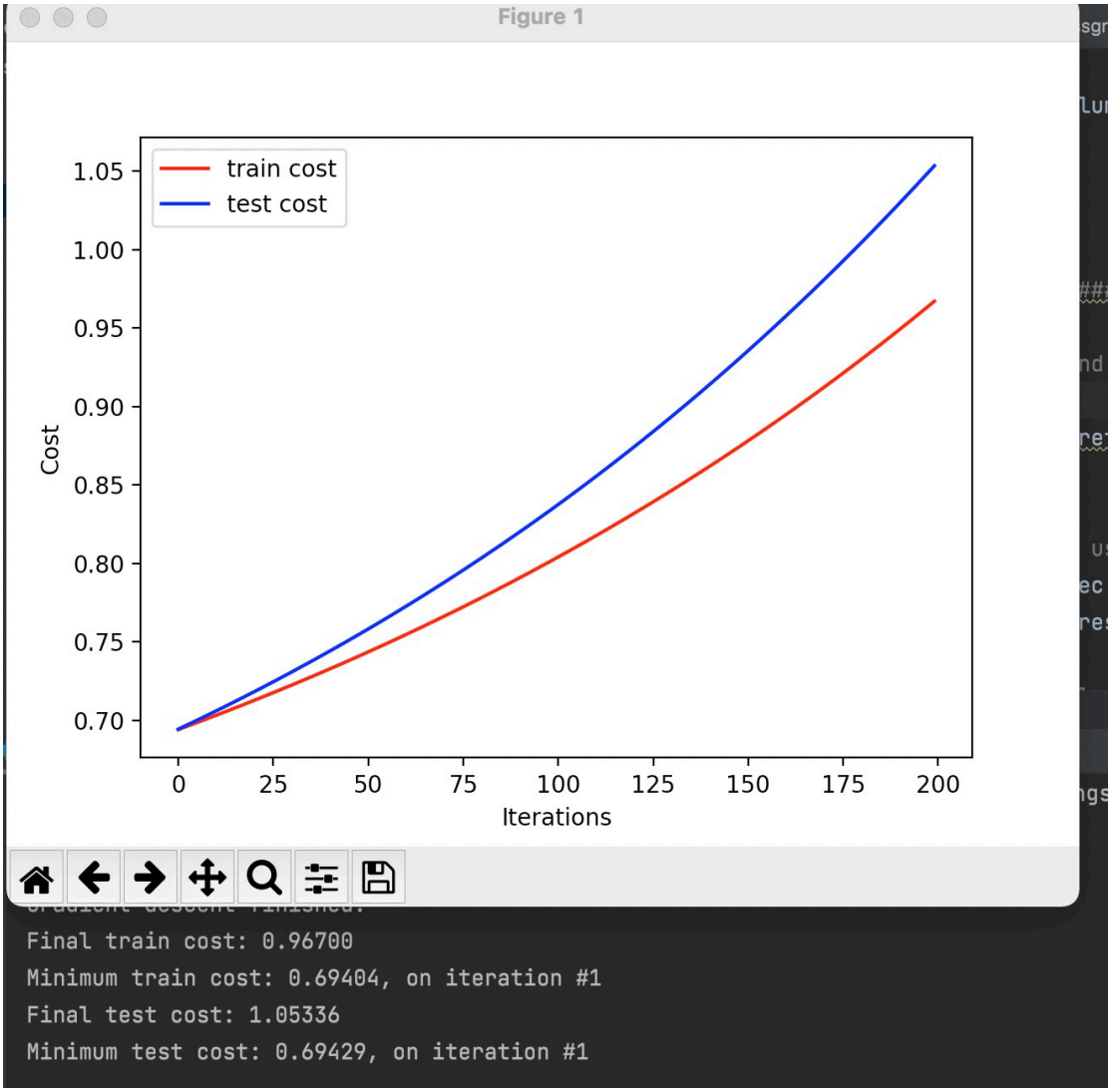
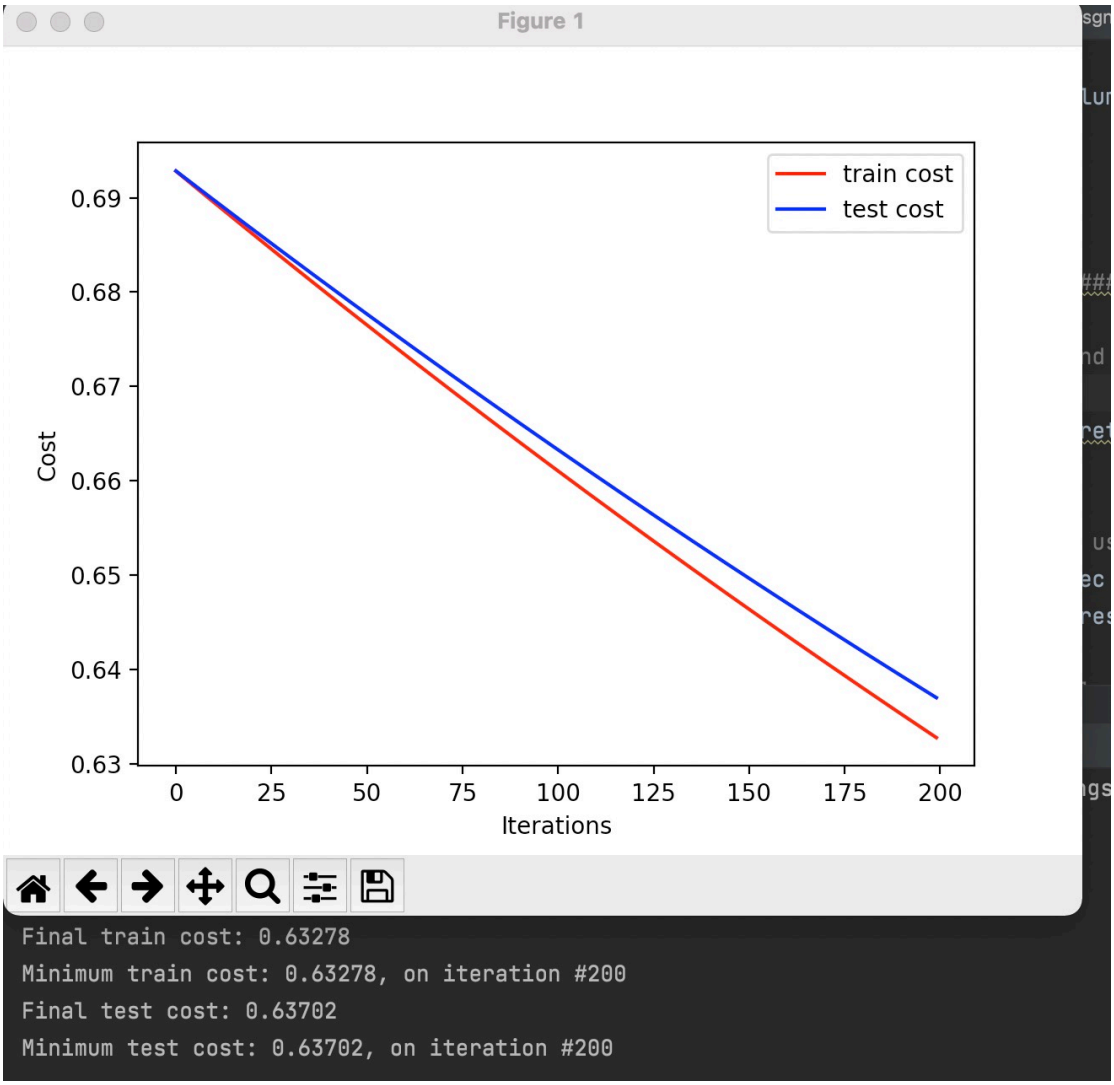


Figure 1
train cost
test cost
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Cost
0
25
50
75
100
125
150
175
200
Iterations
Final train cost: 0.96700
Minimum train cost: 0.69404, on iteration #1
Final test cost: 1.05336
Minimum test cost: 0.69429, on iteration #1

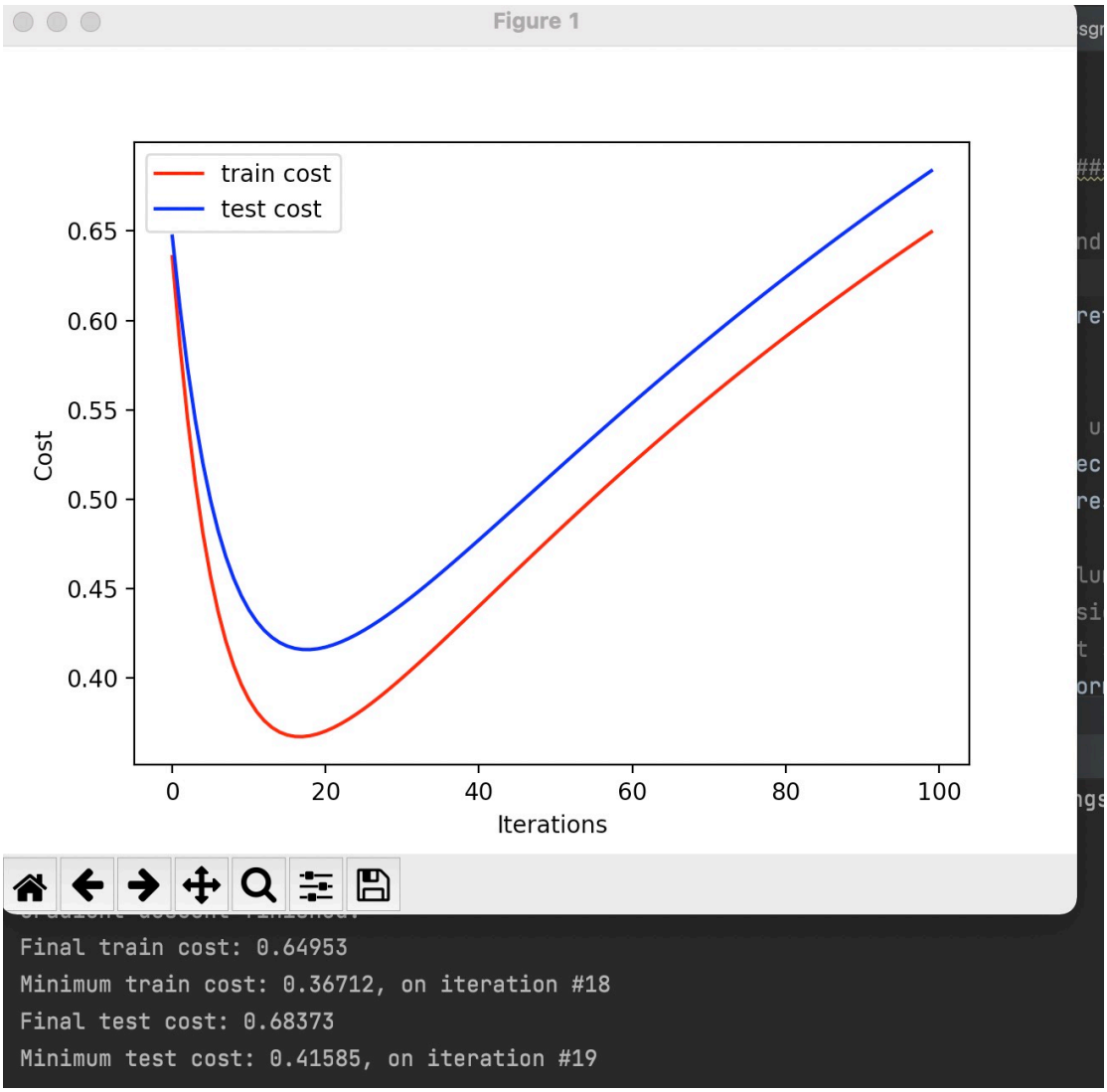
(3) 50-30



Modify assgn1_ex5.py :

```
columns_x1x2 = np.array([], dtype=np.float32)
columns_doublex1 = np.array([], dtype=np.float32)
columns_doublex2 =np.array([], dtype=np.float32)
colums_triplex1 = np.array([],dtype=np.float32)
colums_triplex2 = np.array([],dtype=np.float32)
for i in range(X.shape[0]):
    curren_Value = X[i,0]*X[i,1]
    columns_x1x2 = np.append(columns_x1x2,curren_Value)
    curren_Value = X[i,0]*X[i,0]
    columns_doublex1 = np.append(columns_doublex1,curren_Value)
    curren_Value = X[i,1]*X[i,1]
    columns_doublex2 = np.append(columns_doublex2,curren_Value)
    curren_Value = X[i,0]*X[i,0]*X[i,0]
    colums_triplex1 = np.append(colums_triplex1,curren_Value)
    curren_Value = X[i, 1] * X[i, 1] * X[i, 1]
    colums_triplex2 = np.append(colums_triplex2, curren_Value)
X=np.c_[X,columns_x1x2]
```

```
    curren_Value = X[i,0]*X[i,1]
    columns_x1x2 = np.append(columns_x1x2,curren_Value)
    curren_Value = X[i,0]*X[i,0]
    columns_doublex1 = np.append(columns_doublex1,curren_Value)
    curren_Value = X[i,1]*X[i,1]
    columns_doublex2 = np.append(columns_doublex2,curren_Value)
    curren_Value = X[i,0]*X[i,0]*X[i,0]
    colums_triplex1 = np.append(colums_triplex1,curren_Value)
    curren_Value = X[i, 1] * X[i, 1] * X[i, 1]
    colums_triplex2 = np.append(colums_triplex2, curren_Value)
X=np.c_[X,columns_x1x2]
X=np.c_[X,columns_doublex1]
X=np.c_[X,columns_doublex2]
X=np.c_[X,colums_triplex1]
X=np.c_[X,colums_triplex2]
########################################/
```



When add extra features, the cost value get lower.

Task 9:

It can not create the boundary line that can classifier the two classes accurately.

2. Neural Network

Task 10:

Filling the backward_pass() function :

```
# Step 1. Output deltas are used to update the weights of the output layer
output_deltas = np.zeros((self.n_out))
outputs = self.y_out.copy()


for i in range(self.n_out):
    ########################################
    # Write your code here
    # compute output_deltas : delta_k = (y_k - t_k) * g'(x_k)
    if self.n_out ==1:
        output_deltas[i] = (outputs[i] - targets) * sigmoid_derivative(outputs[i])
    else:
        output_deltas[i] =(outputs[i]- targets[i])*sigmoid_derivative(outputs[i])
    # output_deltas[i] = ...

    ########################################/
```

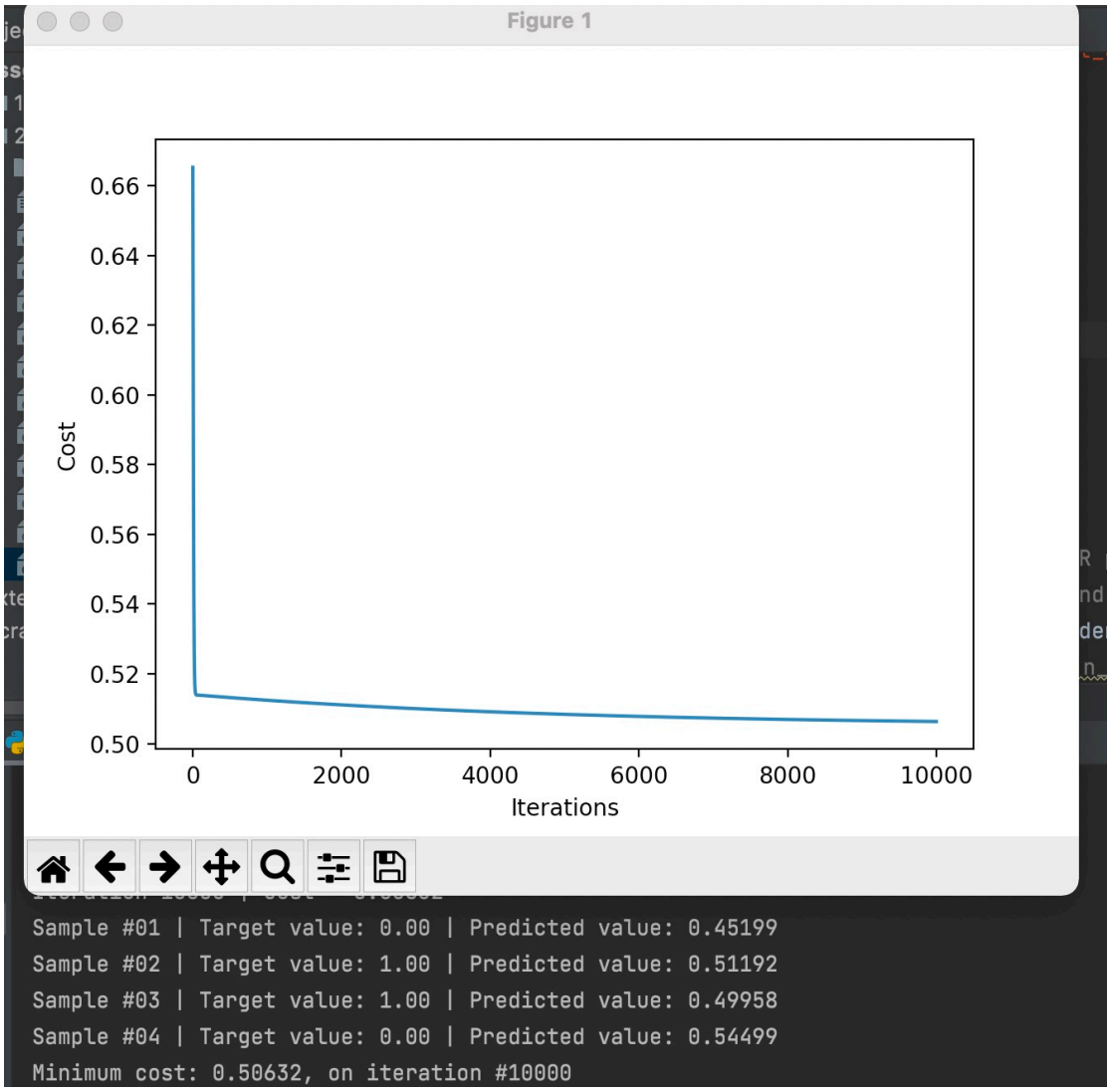
```
# Step 2. Hidden deltas are used to update the weights of the hidden layer
hidden_deltas = np.zeros((len(self.y_hidden)))

# Create a for loop, to iterate over the hidden neurons.
# Then, for each hidden neuron, create another for loop, to iterate over the output neurons
for i in range(len(hidden_deltas)):
    ########################################
    # Write your code here
    # compute hidden_deltas
    sum = 0.0
    for j in range(len(output_deltas)):
        sum = sum + self.w_out[i,j]*hidden_deltas[j]

    hidden_deltas[i] = sigmoid_derivative(self.y_hidden[i]) *sum
    #...
    #...
    #...
```

```
# Step 3. update the weights of the output layer
for i in range(len(self.y_hidden)):
    for j in range(len(output_deltas)):
        ########################################
        # Write your code here
        # update the weights of the output layer

        # self.w_out[i,j] = ...
        self.w_out[i,j] = self.w_out[i,j] - learning_rate * output_deltas[j] * sigmoid(self.y_hidden[i])
        ########################################/

# we will remove the bias that was appended to the hidden neurons, as there is no
# connection to it from the hidden layer
# hence, we also have to keep only the corresponding deltas
hidden_deltas = hidden_deltas[1:]
```

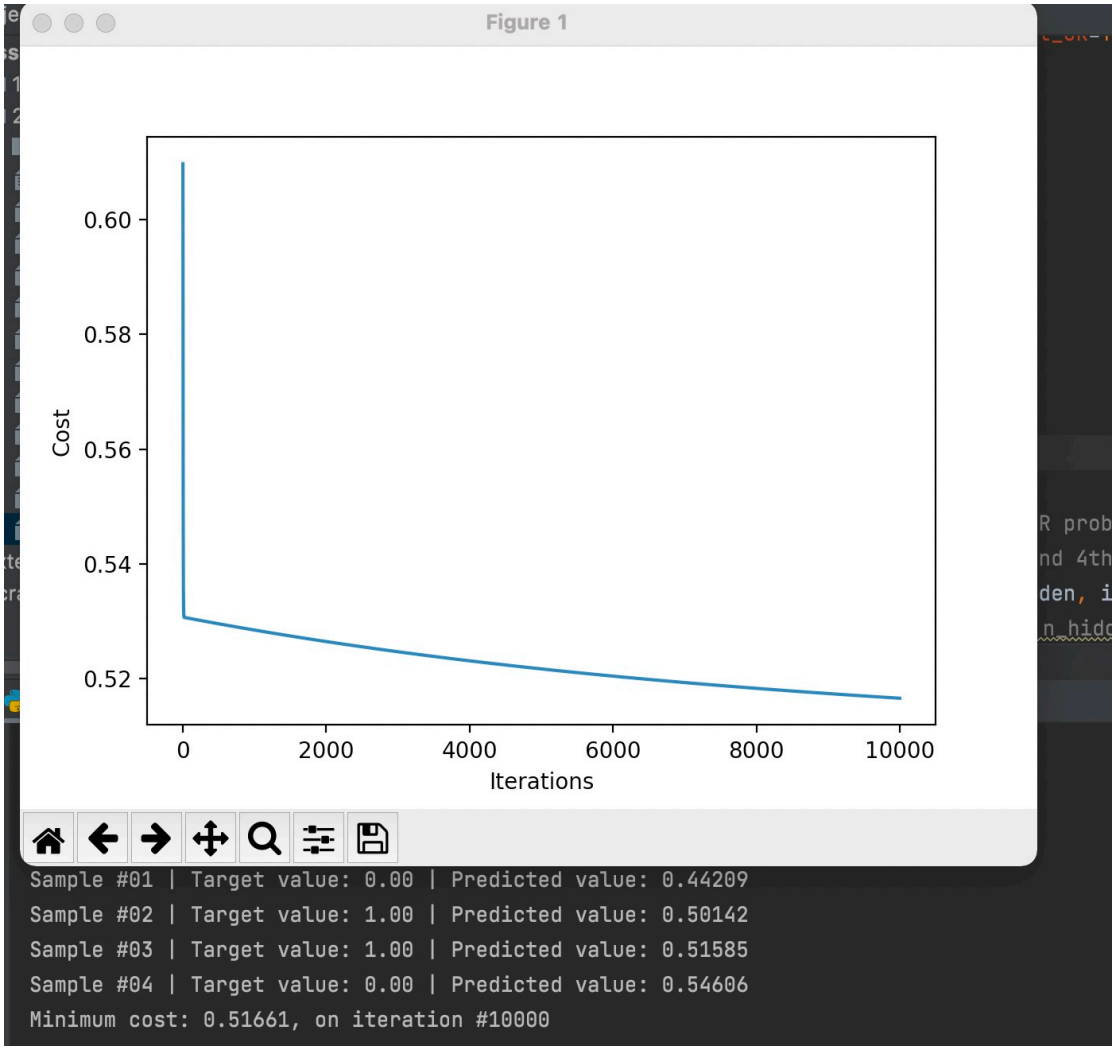
```
# Step 4. update the weights of the hidden layer
# Create a for loop, to iterate over the inputs.
# Then, for each input, create another for loop, to iterate over the hidden deltas
for i in range(len(inputs)):
    for j in range(len(hidden_deltas)):
        ########################################
        # Write your code here
        # update the weights of the hidden layer

        # self.w_hidden[i,j] = ...
        self.w_hidden[i,j] = self.w_hidden[i,j] - learning_rate * hidden_deltas[j] * inputs[i]
        ########################################/

return J
```

Run the file in different learning rate:

Learning rate=0.2:



Learning rate=0.5



Task 11:

Implement a different logical function:

NOR:

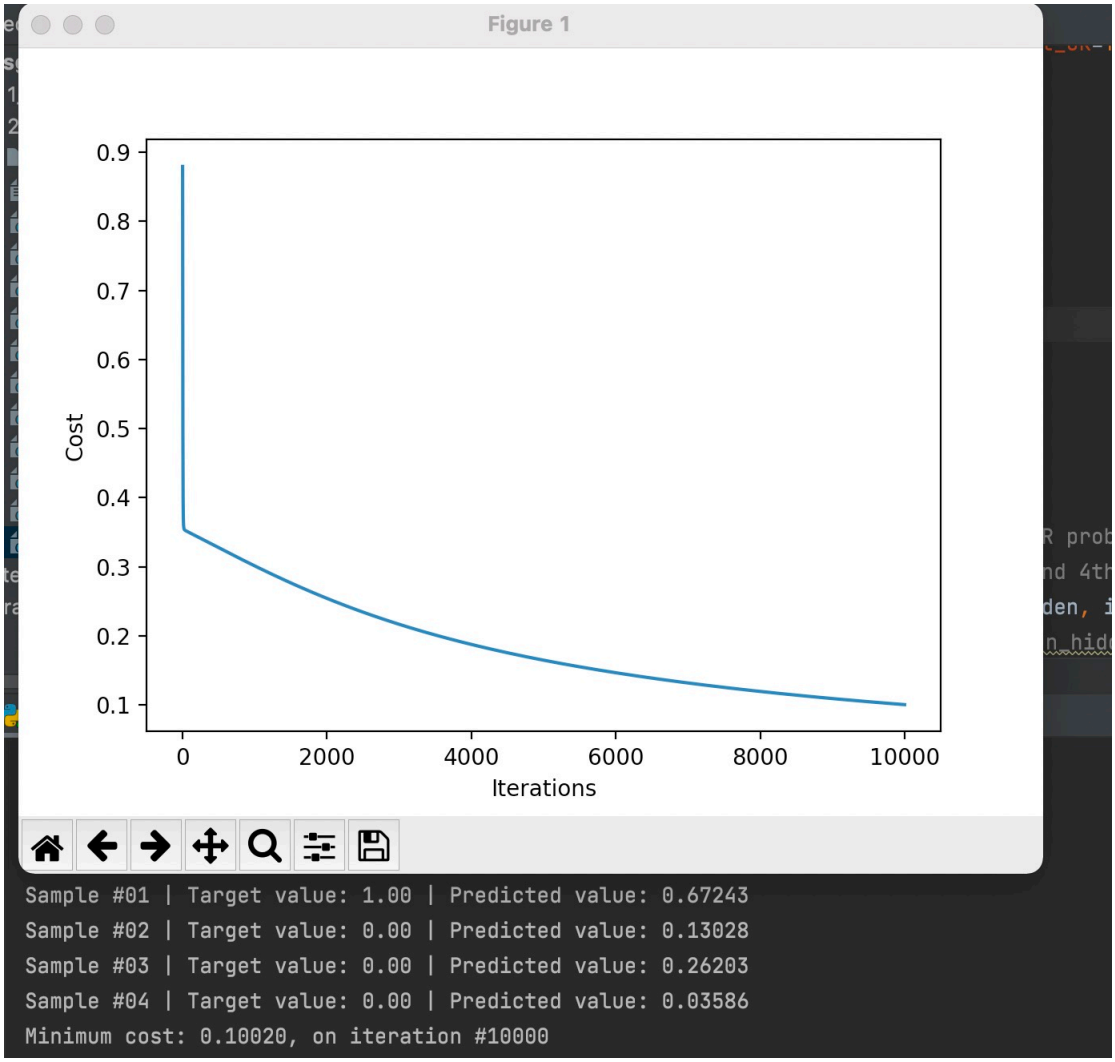
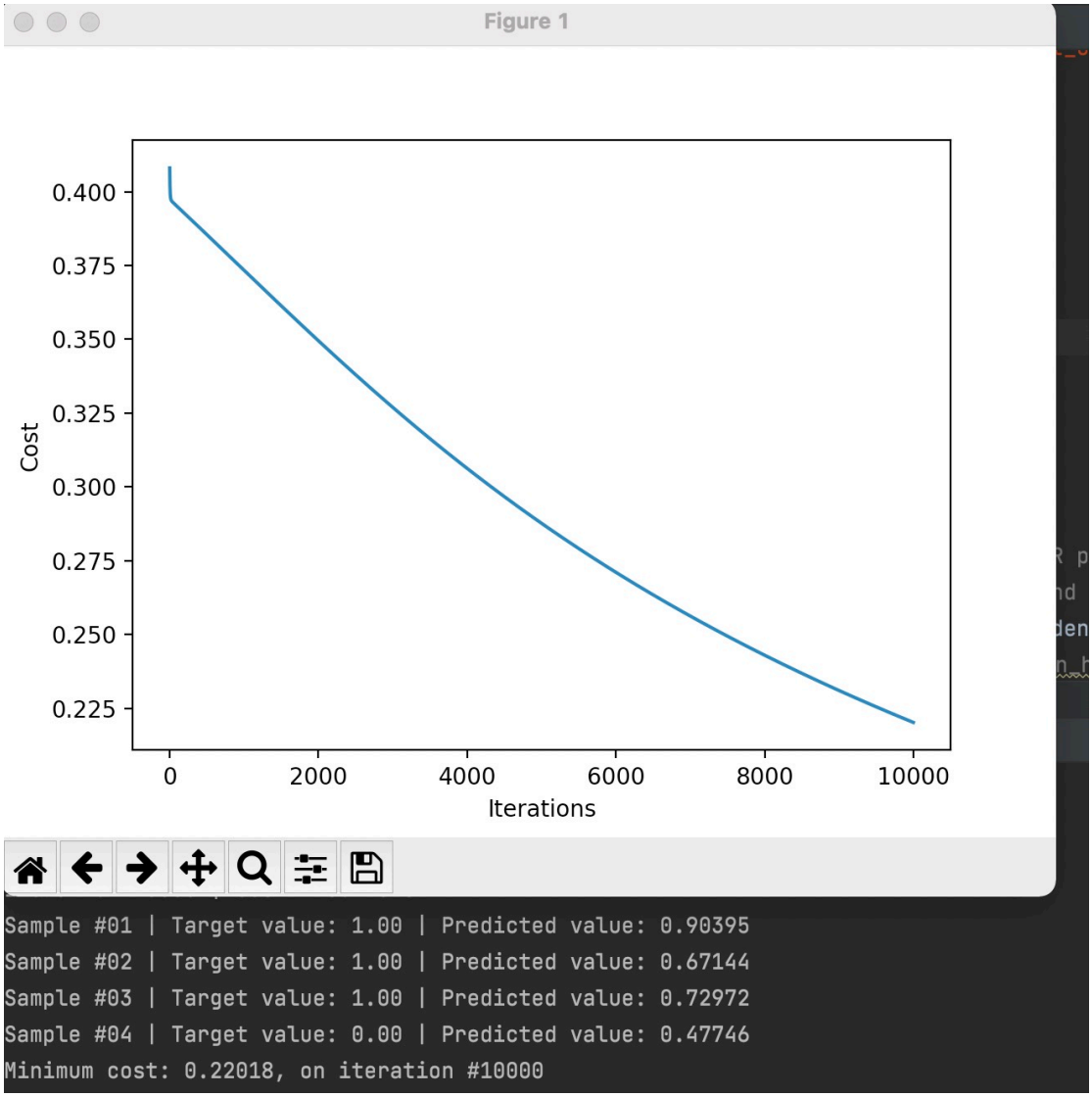


Figure 1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Cost
0
2000
4000
6000
8000
10000
Iterations
Sample #01 | Target value: 1.00 | Predicted value: 0.67243
Sample #02 | Target value: 0.00 | Predicted value: 0.13028
Sample #03 | Target value: 0.00 | Predicted value: 0.26203
Sample #04 | Target value: 0.00 | Predicted value: 0.03586
Minimum cost: 0.10020, on iteration #10000

AND:



Task 12:

Using multinomial logistic regression can achieve the classification. multinomial logistic regression but generalizes it for multiple classes. And using the neural networks can solve the same problem.

Task 13:

Minimum cost: 4.78707, on iteration #100

hidden_number = 1

Minimum cost: 10.58229, on iteration #100

hidden_number = 2

Minimum cost: 8.19815, on iteration #100

hidden_number = 3

Minimum cost: 7.96750, on iteration #100

hidden_number = 5

Minimum cost: 7.77326, on iteration #100

hidden_number = 7

Minimum cost: 7.58255, on iteration #100

hidden_number = 10

Minimum cost: 7.31478, on iteration #100

With the increase of the number of hidden neurons, the cost of the neural net is going down.

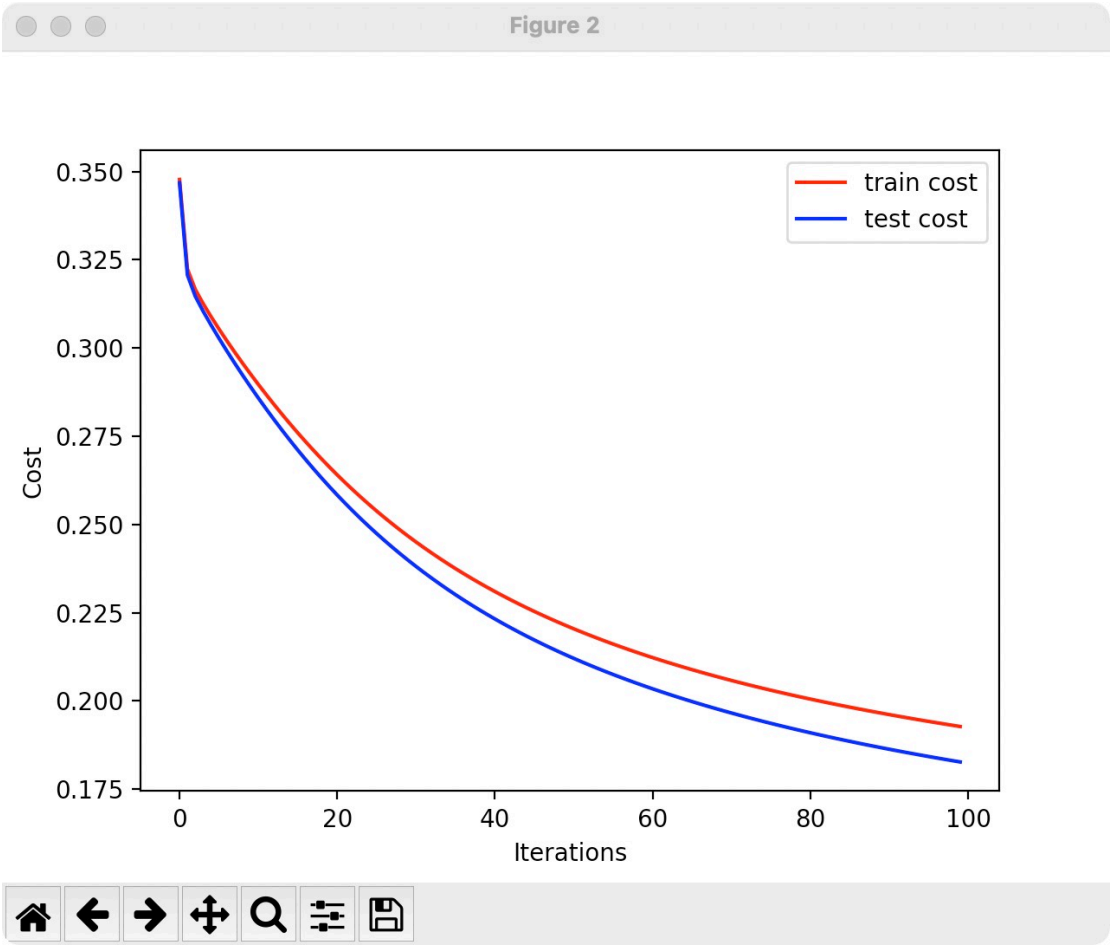


Figure 2
train cost
test cost
Cost
Iterations
0.350
0.325
0.300
0.275
0.250
0.225
0.200
0.175
0
20
40
60
80
100