ECS708 Machine Learning
Assignment 1 – Part 2: Logistic Regression and Neural Networks

1. Logistic Regression

Task 1:

sigmoid.py

```
def sigmoid(z):

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

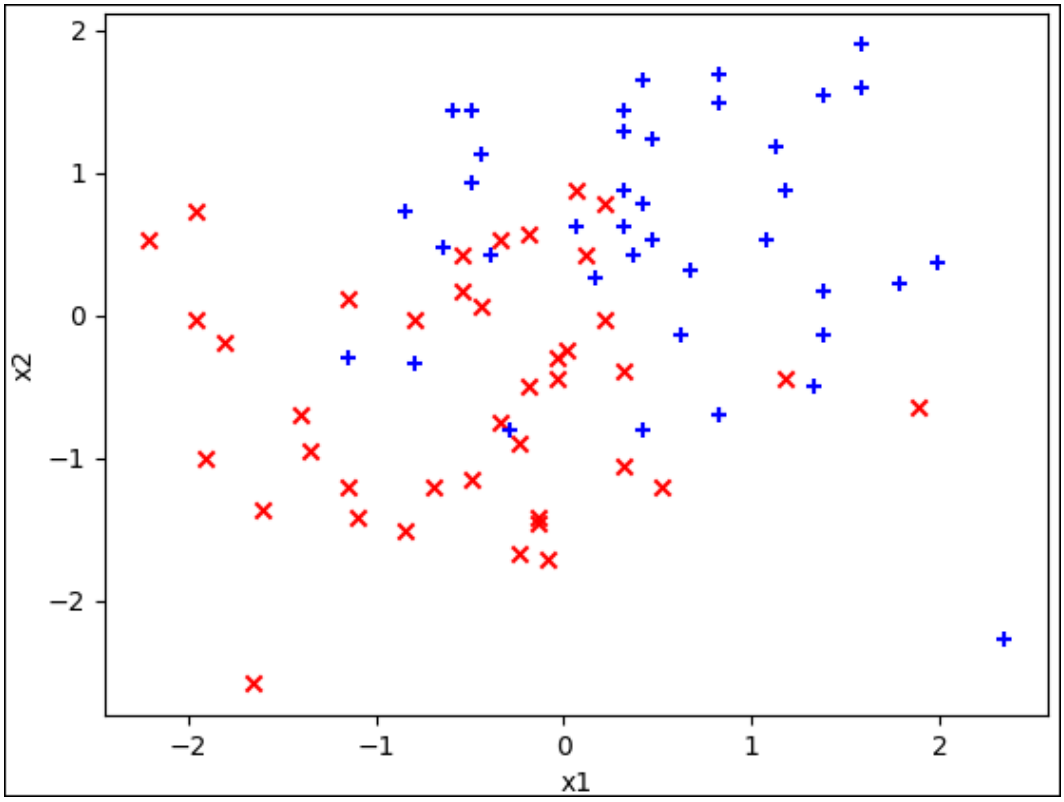


Figure 1: Graph obtained on running plot_data.py

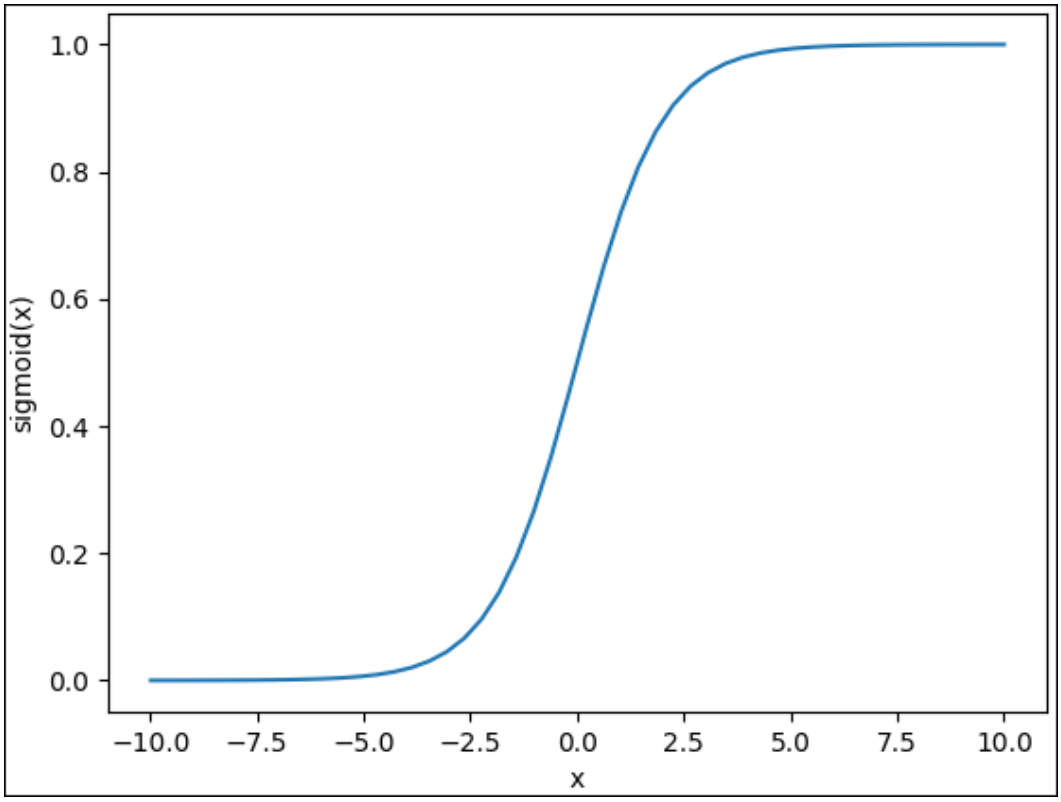
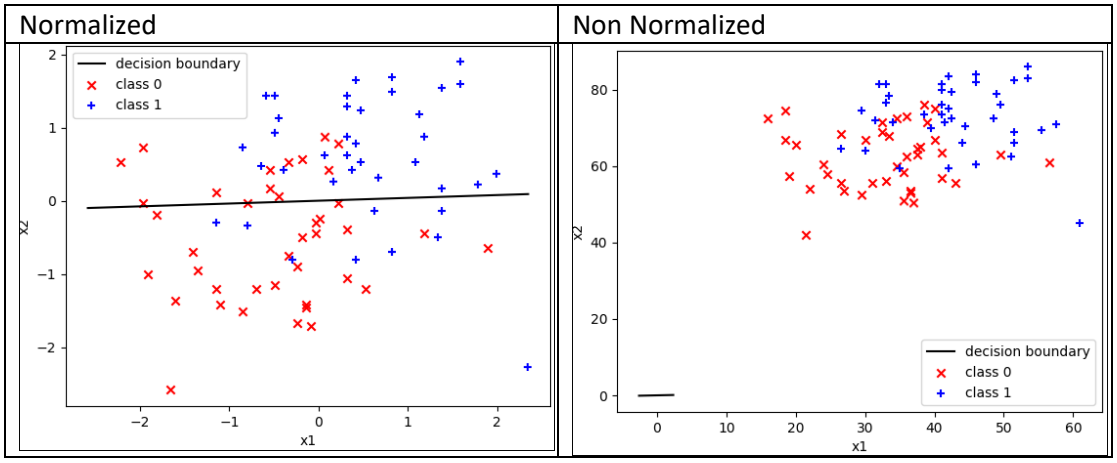


Figure 2: Graph of sigmoid

Task 2:



1.1. Cost function and gradient for logistic regression

Task 3:

calculate_hypothesis.py

```
hypothesis = 0.0
########################################
# Write your code here
# You must calculate the hypothesis for the i-th sample of X, given X, theta and i
for j in range(0, len(theta)):
    hypothesis += theta[j] * (X[i][j])
########################################
result = sigmoid(hypothesis)
```

gradient_descent.py

```
sigma = np.zeros((len(theta)))
for j in range(len(theta)):
    for i in range(m):
        ########################################
        # Write your code here
        # Calculate the hypothesis for the i-th sample of X, with a call to the "calculate_hypothesis" function
        hypothesis = calculate_hypothesis(X, theta, i)
        ########################################
        output = y[i]
        ########################################
        # Write your code here
        # Adapt the code, to compute the values of sigma for all the elements of theta
        sigma[j] = sigma[j] + (hypothesis - output) * X[i, j]
        ########################################
    # update theta_temp
    ########################################
    # Write your code here
    # Update theta_temp, using the values of sigma
    theta_temp = theta_temp - (alpha / m) * sigma
    ########################################
# copy theta_temp to theta
theta = theta_temp.copy()
```

ml_assgn1_ex1.py

```
# initialise trainable parameters theta, set learning rate alpha and number of iterations
theta = np.zeros((3))
alpha = 0.7
iterations = 100
```

Task 4:

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
```

ml_assgn1_ex1.py

```
# initialise trainable parameters theta, set learning rate alpha and number of iterations
theta = np.zeros((3))
alpha = 0.75
iterations = 100
```

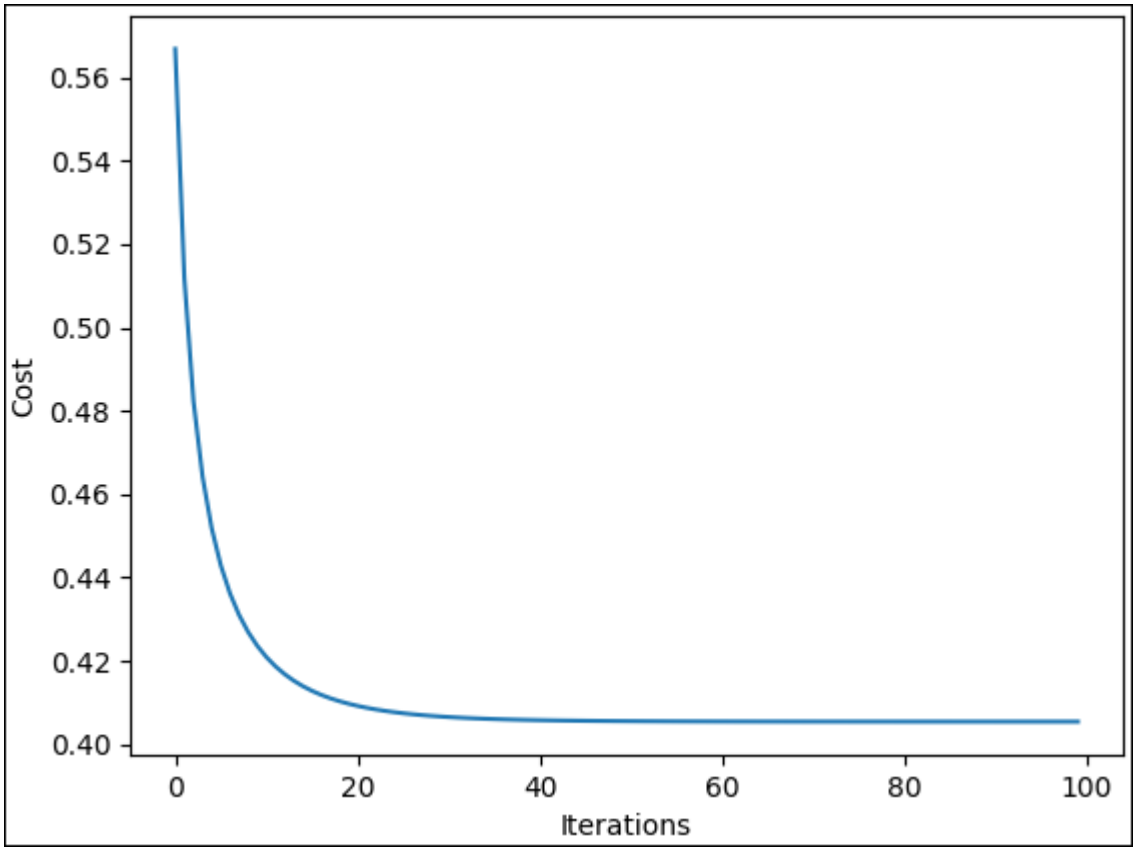


Figure : Graph of cost function for learning rate α = 0.75

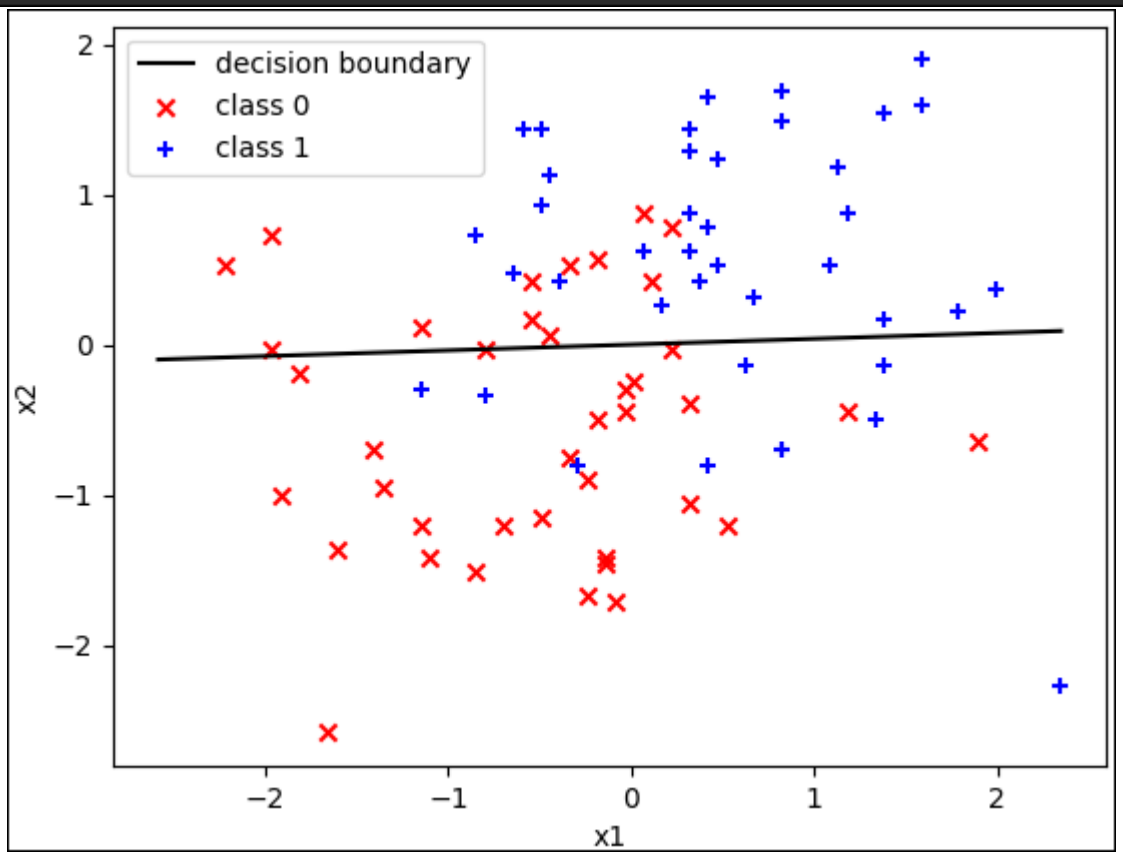
The final cost is 0.40545 for learning rate α = 0.75

1.2. Draw the decision boundary

Task 5:

plot_boundary.py

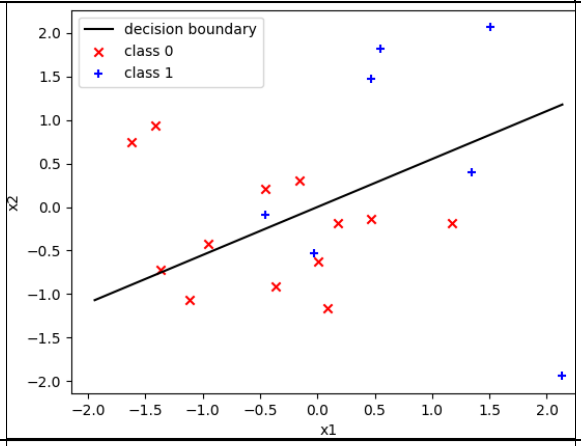
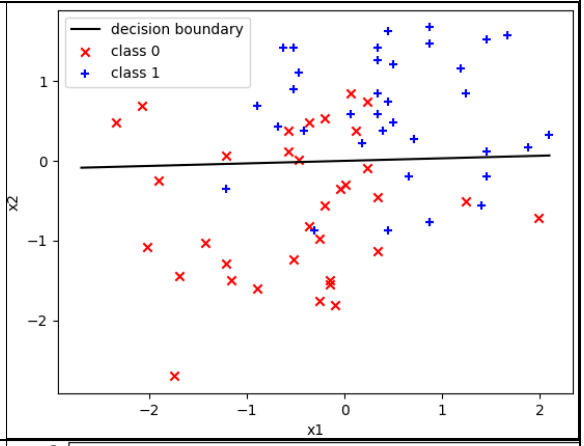
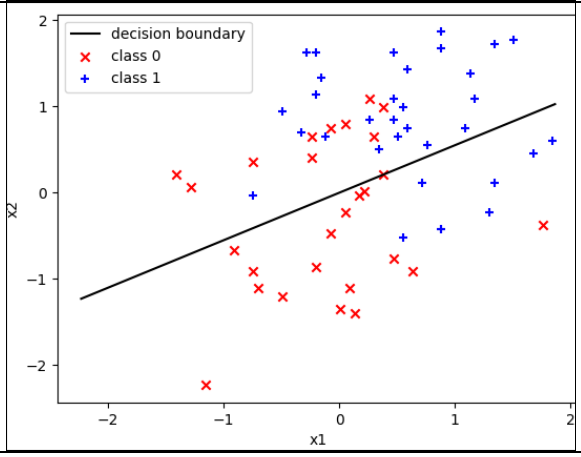
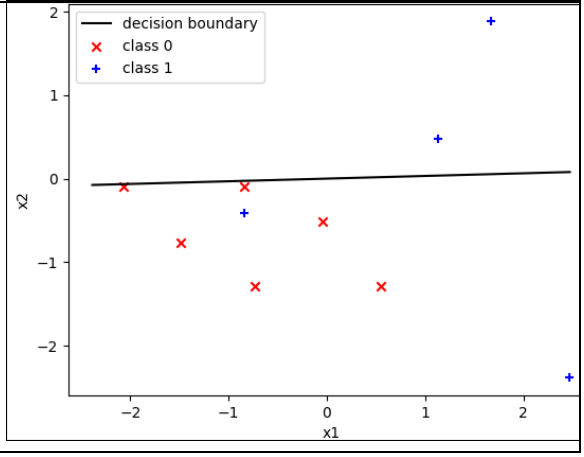
```python
def plot_boundary(X, theta, ax1):

    min_x1 = np.amin(X)
    max_x1 = np.amax(X)
    x2_on_min_x1 = 0.0
    x2_on_max_x1 = 0.0
    ########################################
    # 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 given values of x1
    x2_on_min_x1 = -((theta[0] * min_x1) / theta[1])
    x2_on_max_x1 = -((theta[0] * max_x1) / theta[1])
    ########################################/
    print('min value is: ', min_x1)
    print('max value is: ', max_x1)
    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()
```

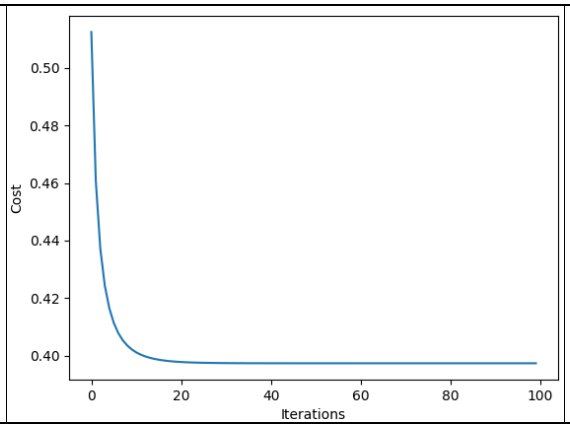
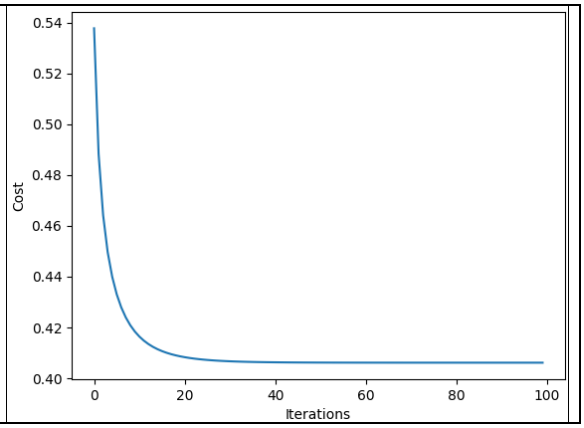


1.3. Non-linear features and overfitting

Task 6:

Sample Size					
20			70		
Cost		Error	Cost		Error
Training	Testing		Training	Testing	
0.30756	0.44121	0.13365	0.41729	0.32665	0.09064
0.40508	0.56053	0.15545	0.40141	0.43673	0.03532
0.28879	0.51942	0.23063	0.41775	0.33234	0.0854
0.34101	0.49635	0.15534	0.38034	0.59444	0.2141
0.24587	0.87676	0.63089	0.39097	0.51465	0.12368
0.32509	0.64552	0.32043	0.42196	0.30856	0.11339
0.33967	0.49104	0.15136	0.40851	0.47388	0.06537
0.33758	0.60407	0.26649	0.38309	0.59211	0.20902
0.32398	0.44245	0.11848	0.40614	0.42882	0.02268
0.39733	0.44237	0.04504	0.41248	0.39011	0.02237

	Sample size: 20	Sample size: 70
Training set		
Testing set		

Cost Function		

The training set generalizes better when the number of samples are higher.
The difference between training cost and test cost is Error.
Above tables show the bad split when sample size is 20 and good split when sample size is 70.

ml_assgn1_ex3.py

```
x1 = X[:, 0]
x2 = X[:, 1]

x1x2 = x1 * x2
x1x2 = np.expand_dims(x1x2, axis=1)
X = np.append(X, x1x2, axis=1)

x12 = x1 ** 2
x12 = np.expand_dims(x12, axis=1)
X = np.append(X, x12, axis=1)

x22 = x2 ** 2
x22 = np.expand_dims(x22, axis=1)
X = np.append(X, x22, axis=1)
```

Task 7:

In task 4, only 2 features are used to train the model and it is evident from the <figure4> that the minimum cost for the machine learning model is 0.40545 and alpha is 0.75. For task 7, we increased the number of features via which machine learning model is getting trained. We added 3 more features into the dataset X. After training the machine learning model with 5 features and with the same learning rate, the minimum cost is coming as 0.403 which is less as compared to previous machine learning model. As we incorporated the non-linear features, hypothesis function as well as weights associated with features are changing and this all is contributing to making the minimum cost is on the lower side.

Task 8:

gradient_descent_training.py

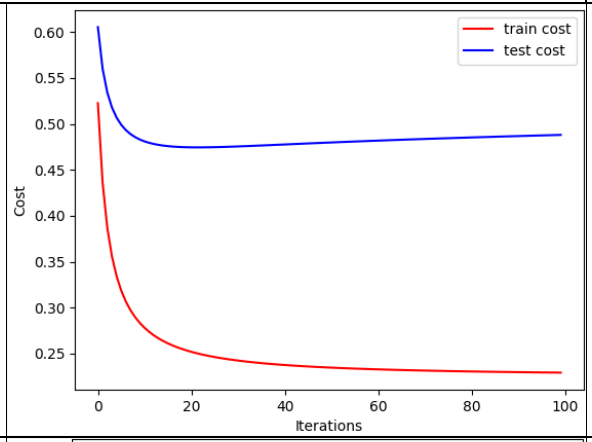
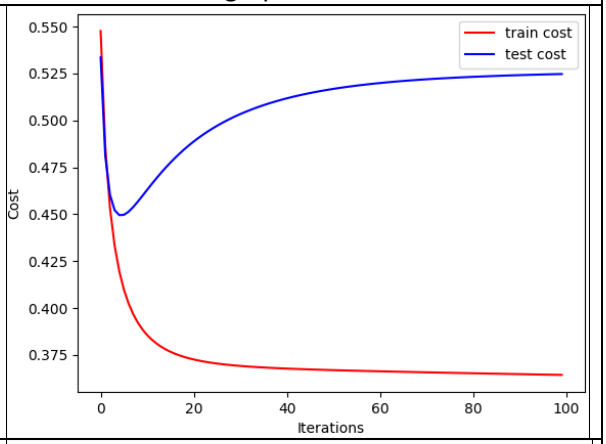
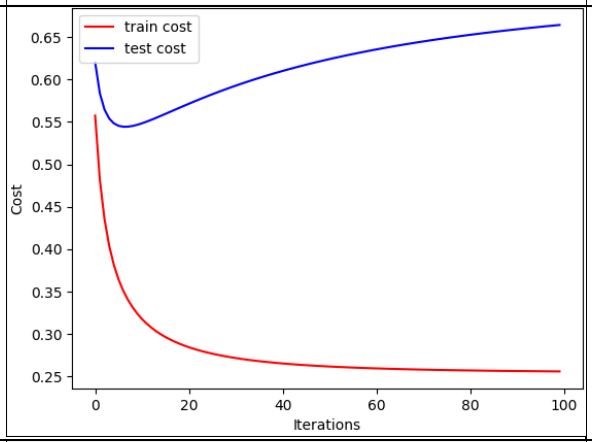
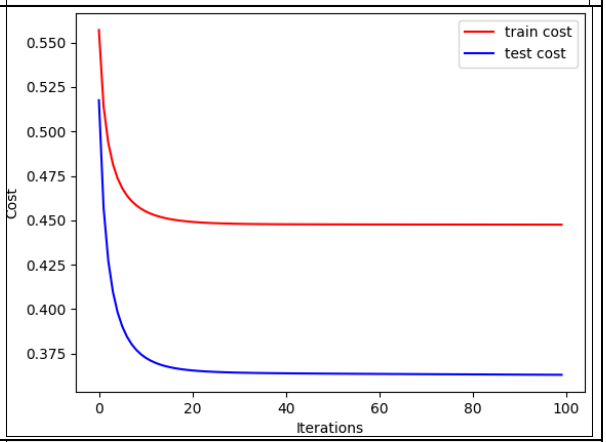
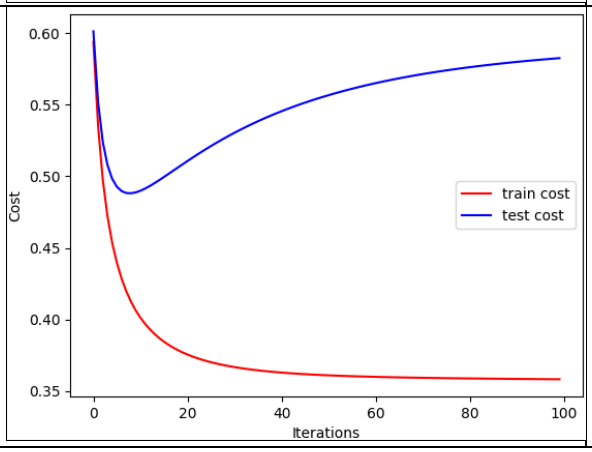
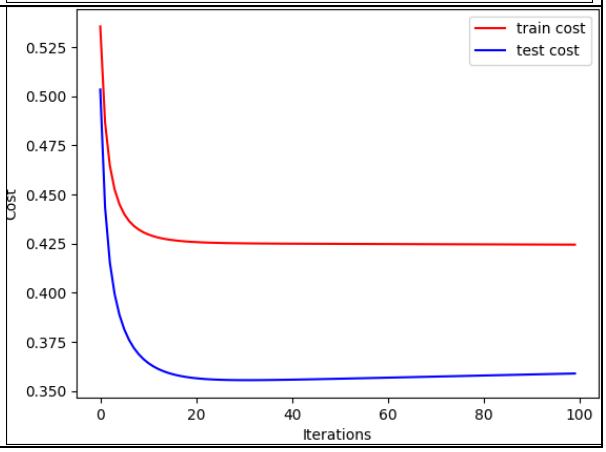
```
sigma = np.zeros((len(theta)))
for j in range(len(theta)):
    for i in range(m):
        ########################################
        # Write your code here
        # Calculate the hypothesis for the i-th sample of X, with a call to the "calculate_hypothesis" function
        hypothesis = calculate_hypothesis(X_train, 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[j] = sigma[j] + (hypothesis - output) * X_train[i, j]
        ########################################/
    # update theta_temp
    ########################################
    # Write your code here
    # Update theta_temp, using the values of sigma
    theta_temp = theta_temp - (alpha / m) * sigma
    ########################################/
###############
# copy theta_temp to theta
theta = theta_temp.copy()
```

ml_assgn1_ex4.py

```
x1 = X[:, 0]
x2 = X[:, 1]

x1x2 = x1 * x2
x1x2 = np.expand_dims(x1x2, axis=1)
X = np.append(X, x1x2, axis=1)

x12 = x1 ** 2
x12 = np.expand_dims(x12, axis=1)
X = np.append(X, x12, axis=1)

x22 = x2 ** 2
x22 = np.expand_dims(x22, axis=1)
X = np.append(X, x22, axis=1)
```

ml_assgn1_ex5.py

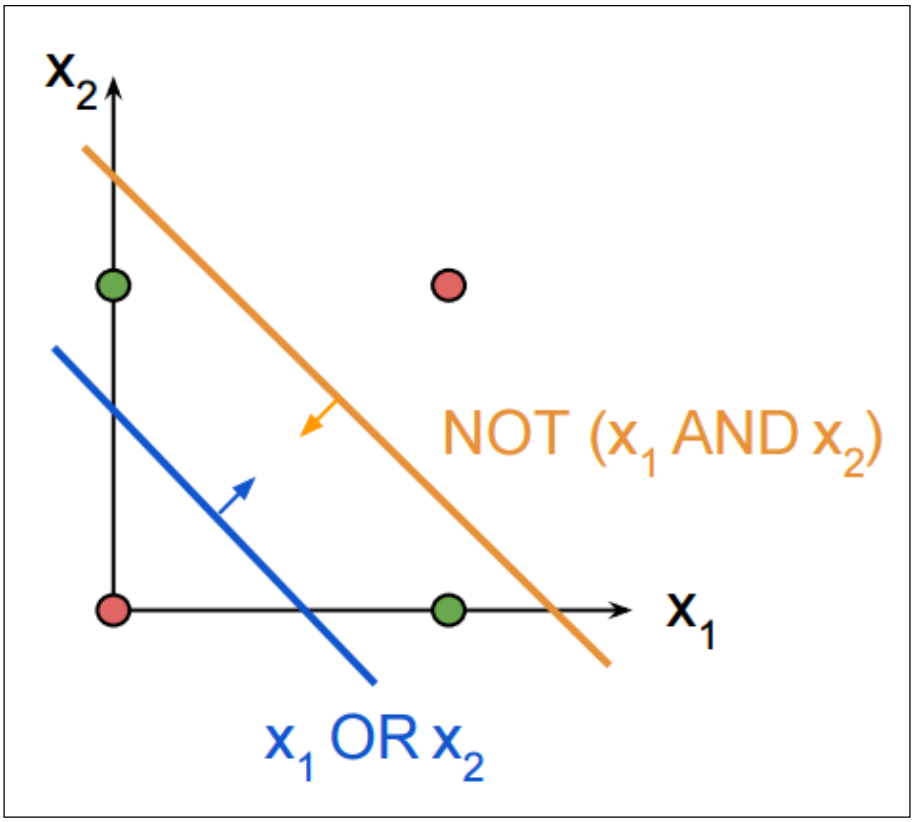
```
x1 = X[:, 0]
x2 = X[:, 1]

x1x2 = x1*x2
x1x2 = np.expand_dims(x1x2, axis=1)
X = np.append(X, x1x2, axis=1)

x12 = x1**2
x12 = np.expand_dims(x12, axis=1)
X = np.append(X, x12, axis=1)

x22 = x2**2
x22 = np.expand_dims(x22, axis=1)
X = np.append(X, x22, axis=1)

x13 = x1**3
x13 = np.expand_dims(x13, axis=1)
X = np.append(X, x13, axis=1)

x23 = x2**3
x23 = np.expand_dims(x23, axis=1)
X = np.append(X, x23, axis=1)
```

	Primary Feature Graph	7 Features Vector graph
Sample Size = 20		
Sample Size = 40		
Sample Size = 60		

Task 9:

Truth table for XOR

Input1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0



In logistic regression, the decision boundary is plotted to classify the dataset into binary classes as 0 and 1. For XOR, there cannot be a single decision boundary for classifying both the classes

2. Linear Regression with Multiple Variables

Task 10:

NeuralNetwork.py

```
# 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 np.isscalar(targets):
        output_deltas[i] = (outputs[i] - targets) * sigmoid_derivative(outputs[i])
    else:
        output_deltas[i] = (outputs[i] - targets[i]) * sigmoid_derivative(outputs[i])
```

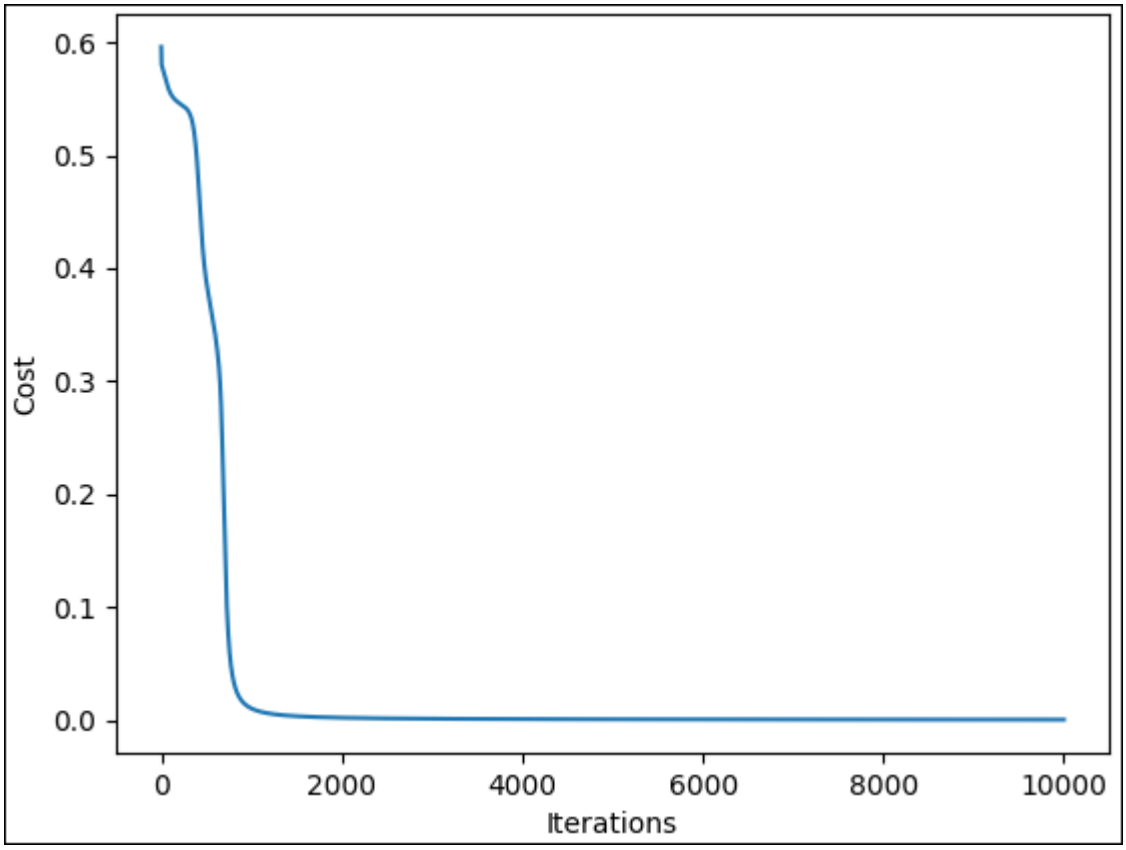
```
hidden_deltas = np.zeros((len(self.y_hidden)))

# 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)):
    summation = 0
    ########################################
    # Write your code here
    # compute hidden_deltas
    for j in range(len(output_deltas)):
        summation += self.w_out[i, j] * output_deltas[j]
    hidden_deltas[i] = sigmoid_derivative(self.y_hidden[i]) * summation
```

```
# 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] -= (learning_rate * output_deltas[j] * 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] -= (learning_rate * hidden_deltas[j] * inputs[i])
```

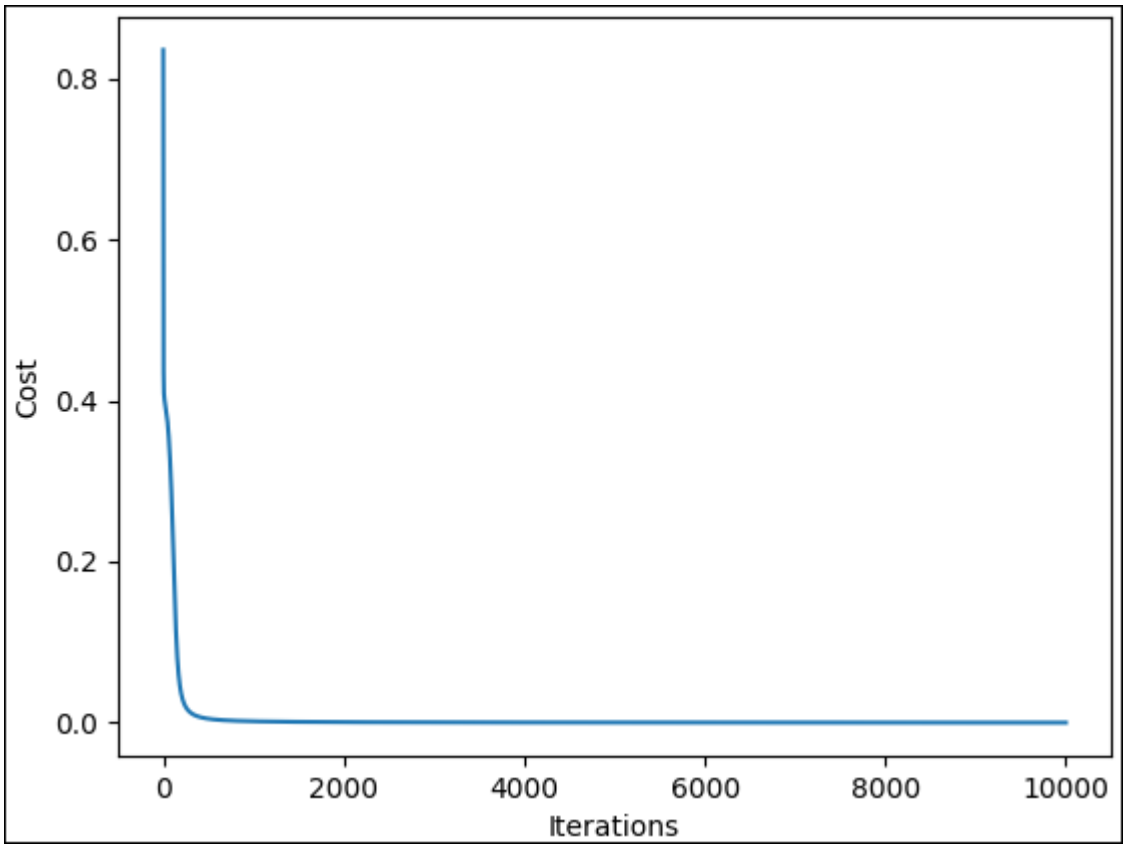
```
Sample #01 | Target value: 0.00 | Predicted value: 0.00982
Sample #02 | Target value: 1.00 | Predicted value: 0.98978
Sample #03 | Target value: 1.00 | Predicted value: 0.98971
Sample #04 | Target value: 0.00 | Predicted value: 0.01260
Minimum cost: 0.00023, on iteration #10000
```

Minimum cost is 0.00023 at alpha = 1.2



Task 11:

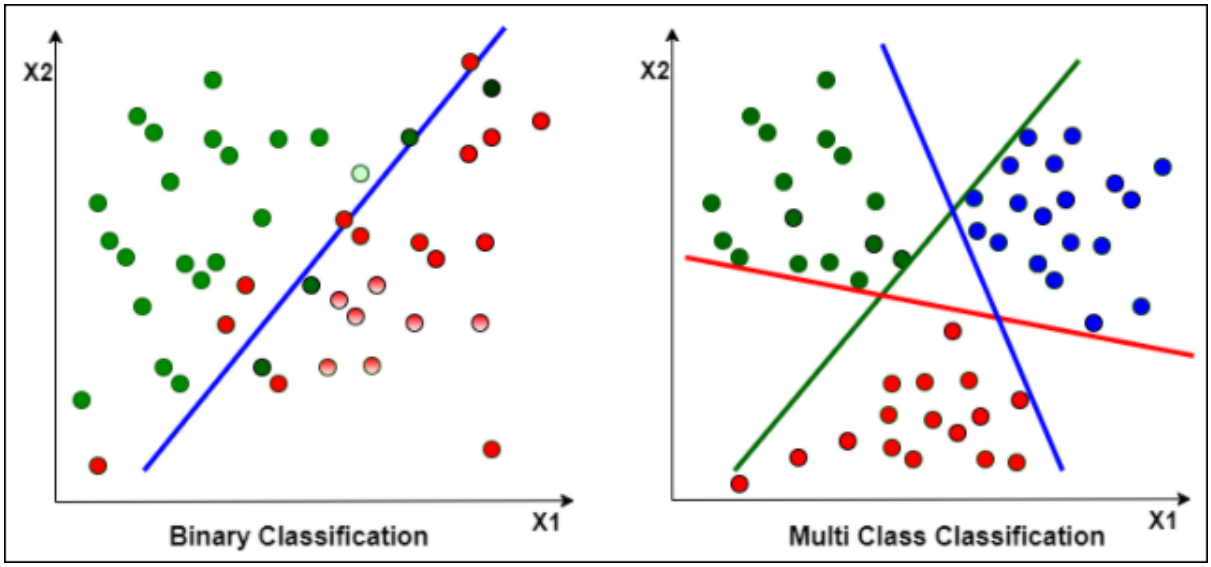
The following results are for **AND** gate



```
Sample #01 | Target value: 0.00 | Predicted value: 0.00153
Sample #02 | Target value: 0.00 | Predicted value: 0.00707
Sample #03 | Target value: 0.00 | Predicted value: 0.00707
Sample #04 | Target value: 1.00 | Predicted value: 0.98790
Minimum cost: 0.00012, on iteration #10000
```

Minimum cost is 0.00012 at alpha = 1.2

Task 12:



When using logistic regression for multi class classification, we treat pair of classes and draw decision boundary between them. Hence for 3 classes we will have 3 pairs and 3 decision boundaries. Ambuiguity arises when the data point will lie in the intersection of 3 boundaries.

Task 13:

Alpha = 0.4

Hidden layer	Minimum cost
1	3.25795
2	0.14930
3	0. 14123
5	0.13025
7	0.13874
9	0.14908
10	0.14967

For alpha = 0.4 hidden no. of neurons should be 3 and 5