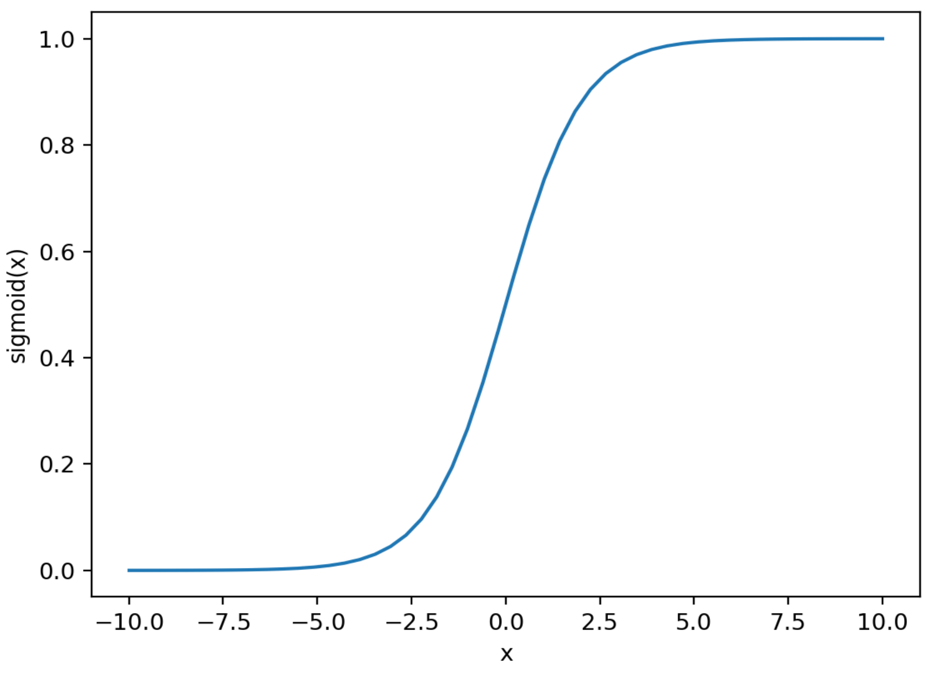
Assignment 1 – Part 2: Logistic Regression and Neural Networks

1. Logistic Regression

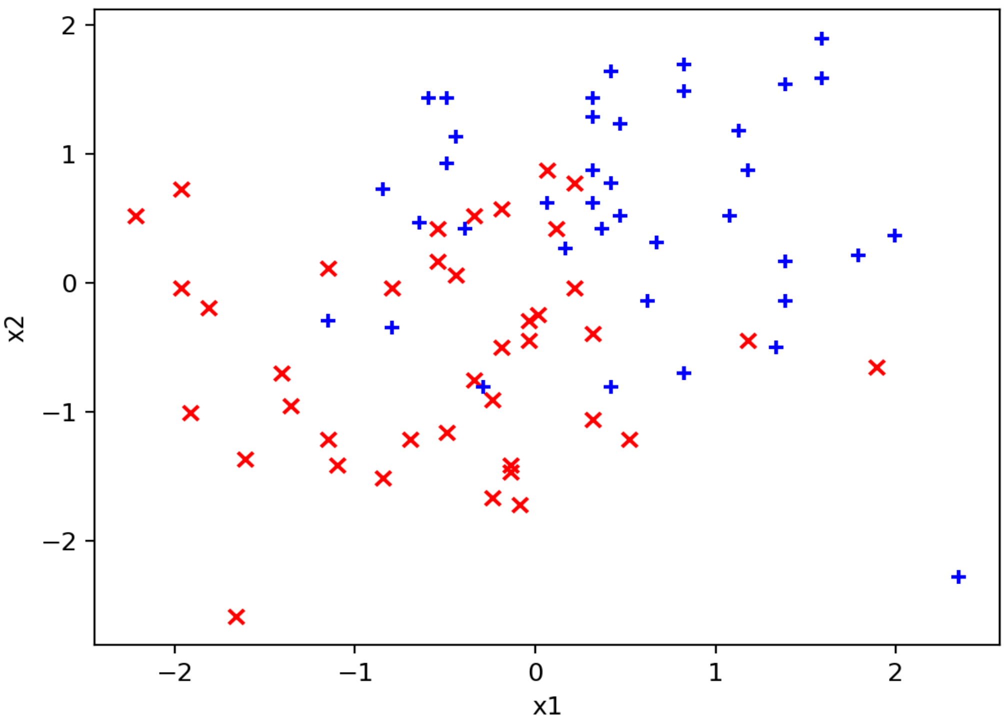
The formula for the given hypothesis in logistic regression is a logistic/sigmoid given below:

```
output = 1 / (1 + np.exp(-z))

return output
```

After plotting the plot_sigmoid.py function to plot the sigmoid function is given below:



Note: After running the plot_data.py we can clearly observe that the da ta have been normalised, to enable the data to be more easily optimised.



In plot_boundary I am rearranging the terms in the equation of the hypothesis function.

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

    a = X[..., 1]
    b = X[..., 2]

    min_x1 = np.max(a)
    max_x1 = np.min(a)
    ia = np.where(a == min_x1)
    ib = np.where(a == max_x1)
    x2_on_min_x1 = b[ia]
    x2_on_max_x1 = b[ib]
```

```
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')
```

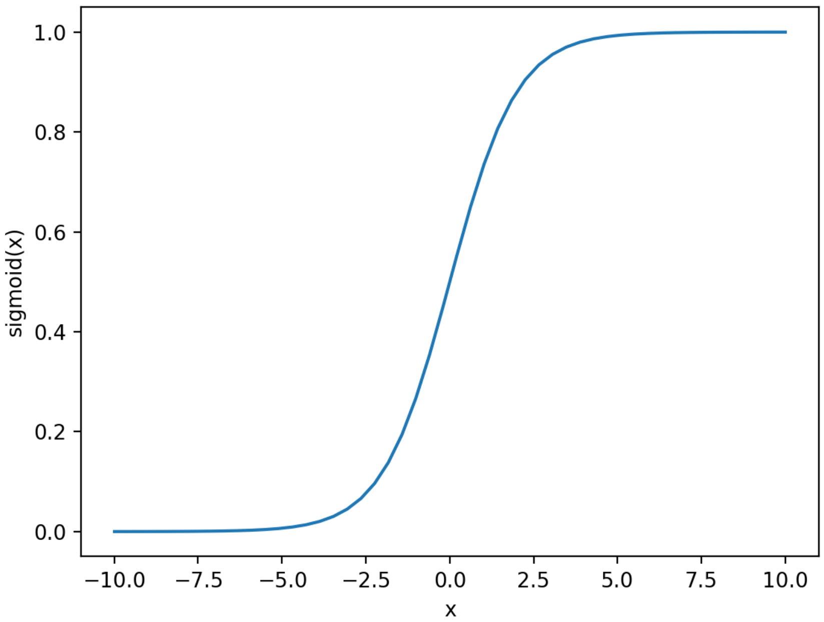
plots are given below after running the ml_assgn1_ex1 for

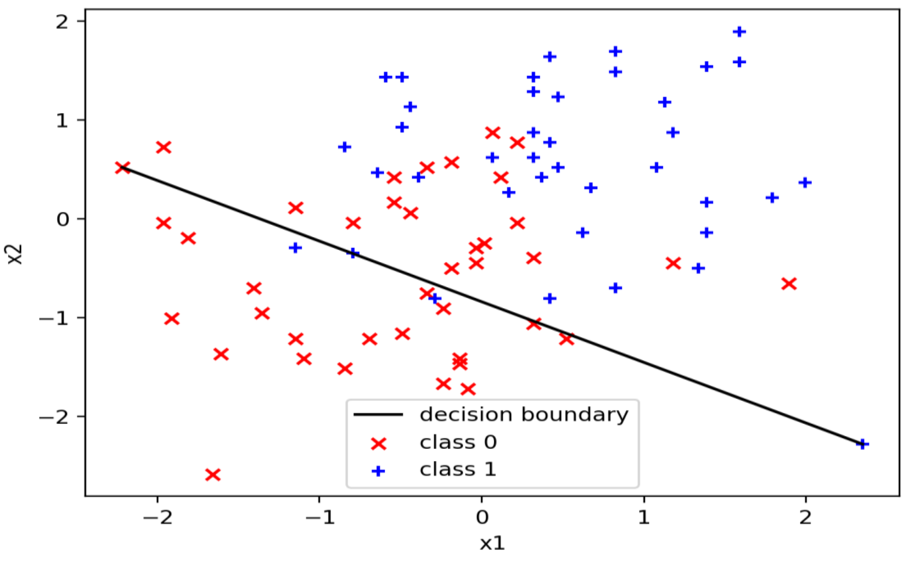
```
alpha = 0.024
iterations = 100
```

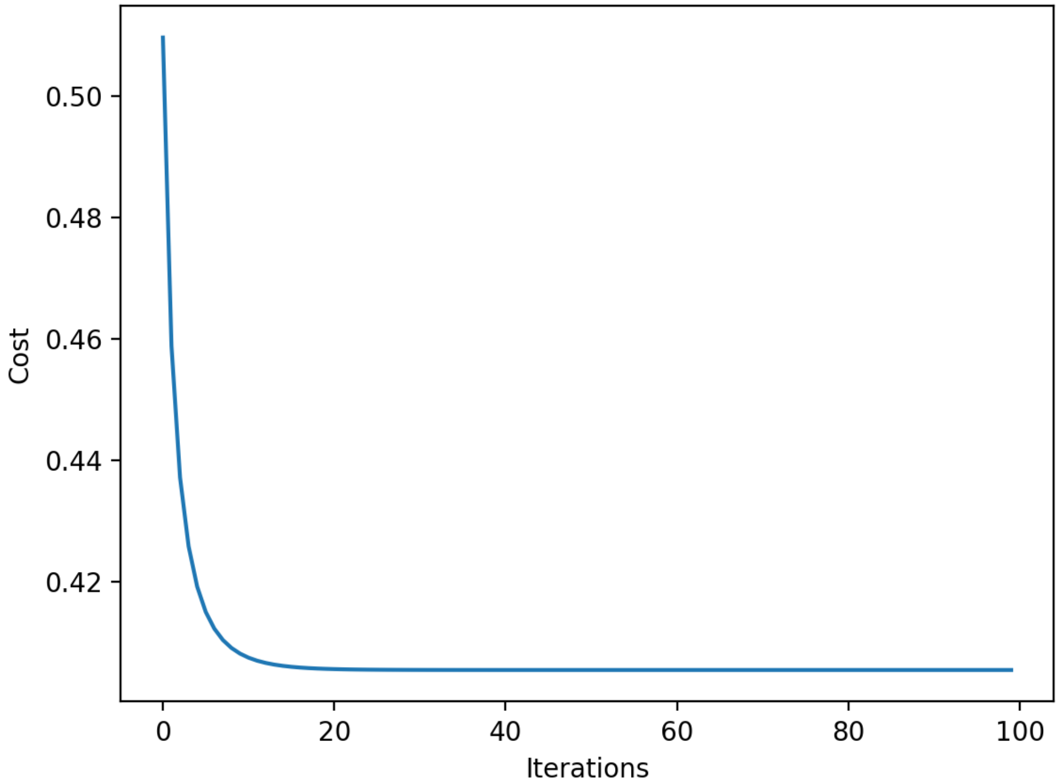
```
cost = (-output)*np.log(hypothesis) - (1 -
output) * np.log(1-hypothesis)
```

For training sample 20

Final training cost: 0.17416
Minimum training cost: 0.17416, on iteration #100
Final test cost: 0.73745

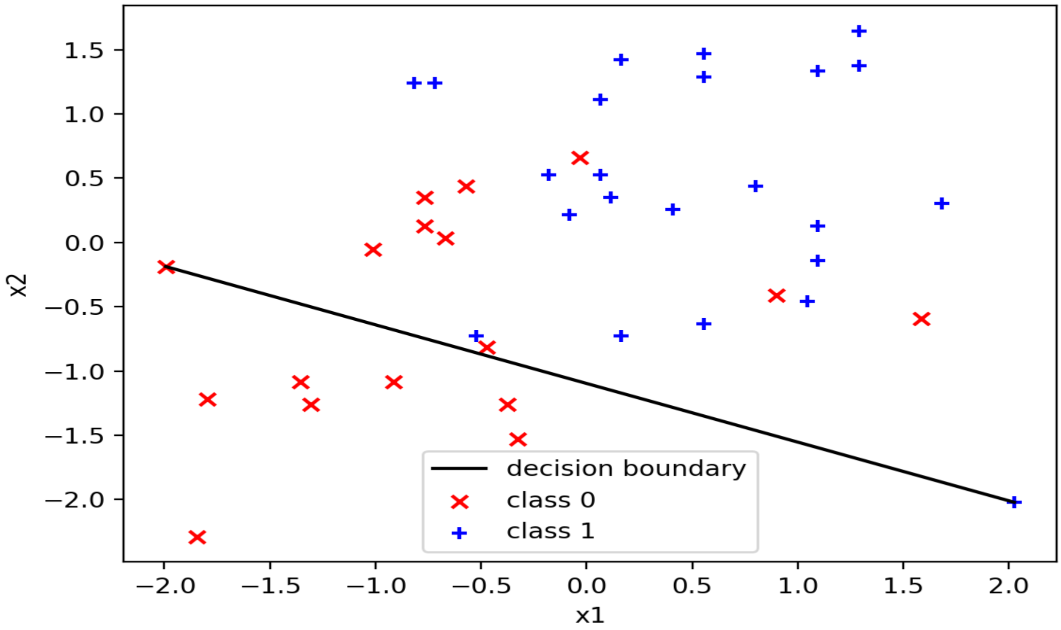


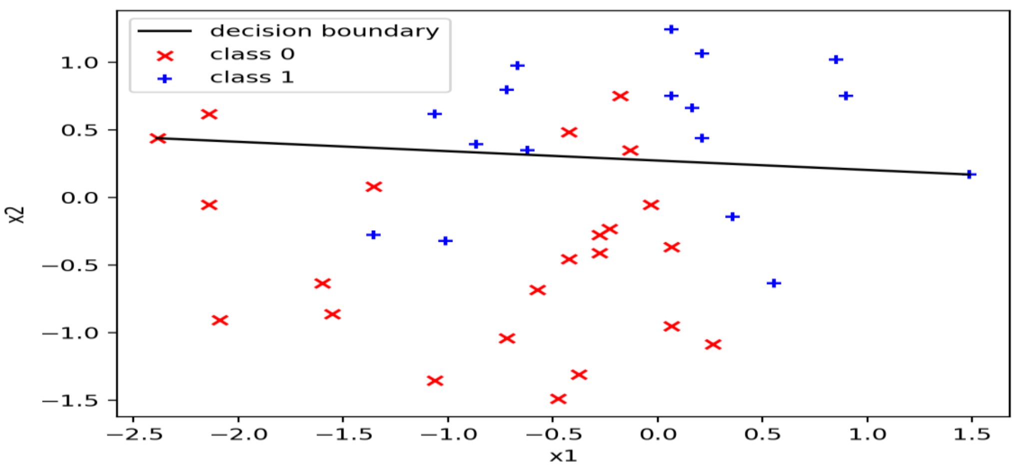


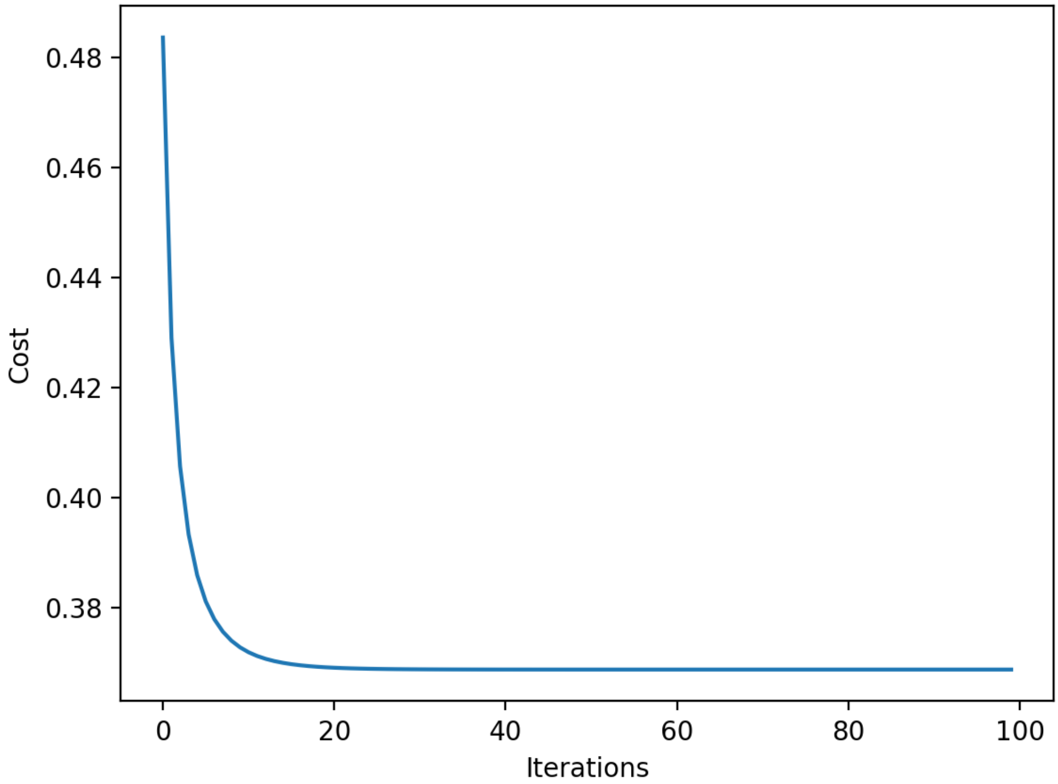


For training sample 40

Final training cost: 0.41818
Minimum training cost: 0.41818, on iteration #100
Final test cost: 0.39502


decision boundary
class 0
class 1
x1
x2
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

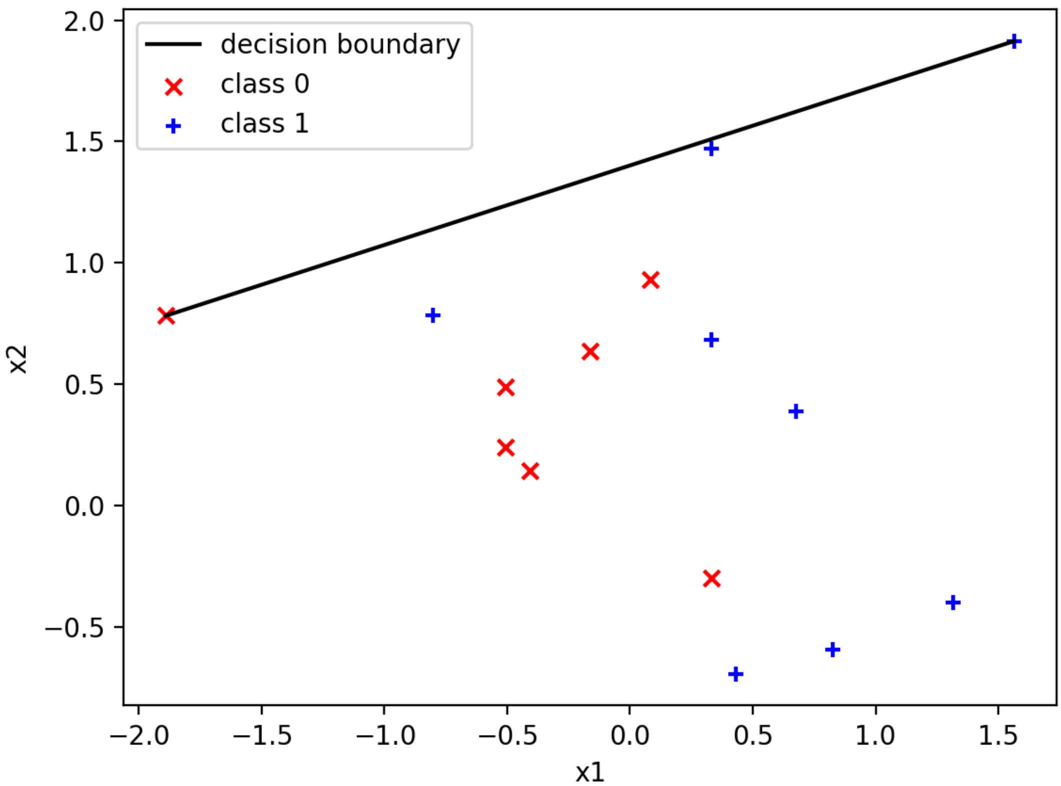

decision boundary
class 0
class 1
x1
x2
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

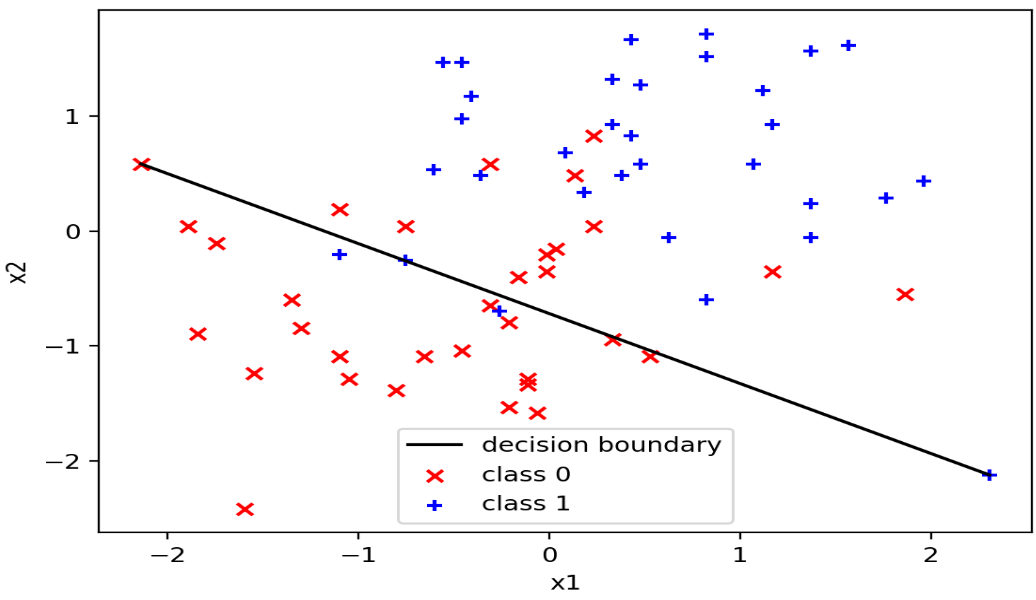


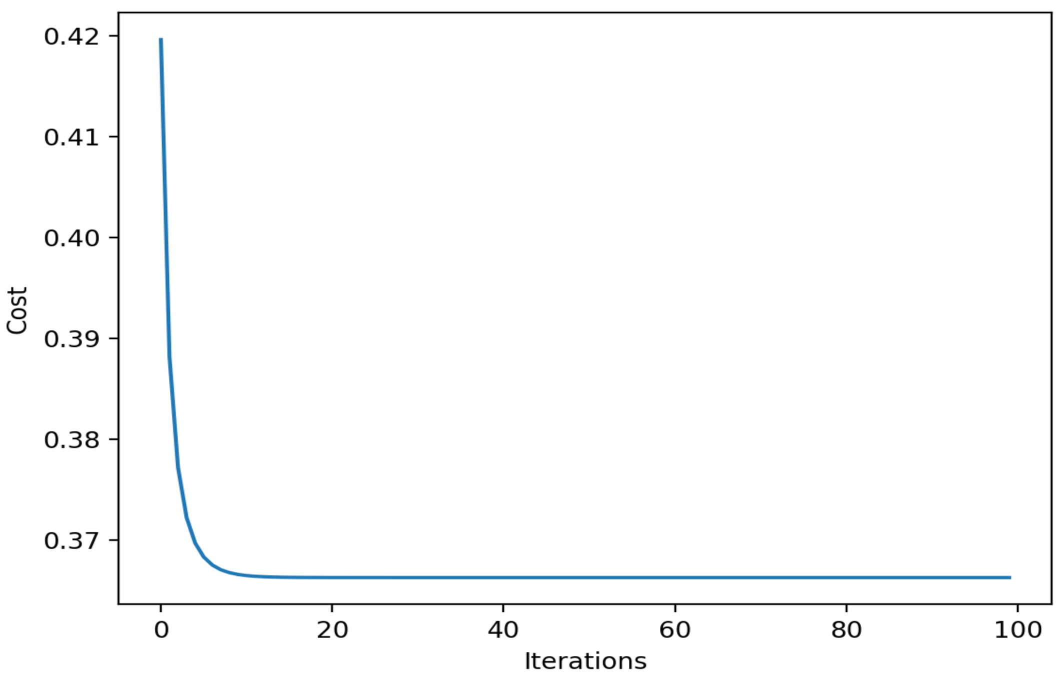
0.48
0.46
0.44
0.42
0.40
0.38
Cost
0
20
40
60
80
100
Iterations

For sample 65

Final training cost: 0.42026
Minimum training cost: 0.42026, on iteration #60
Final test cost: 0.02577

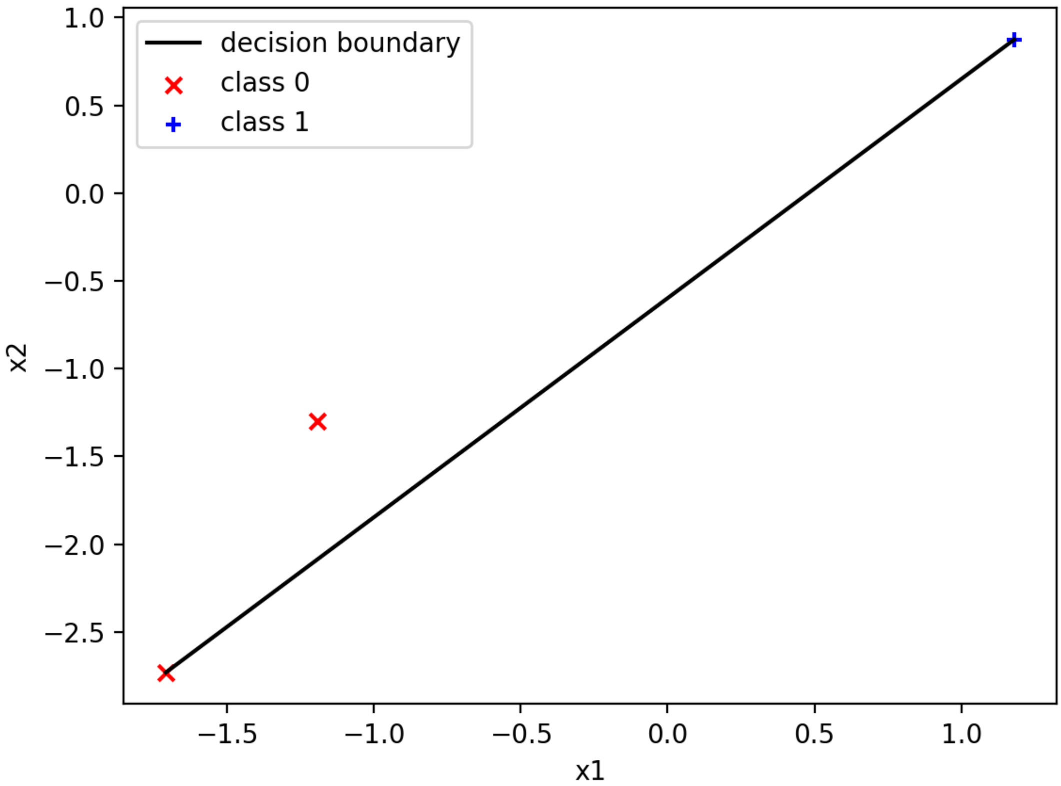


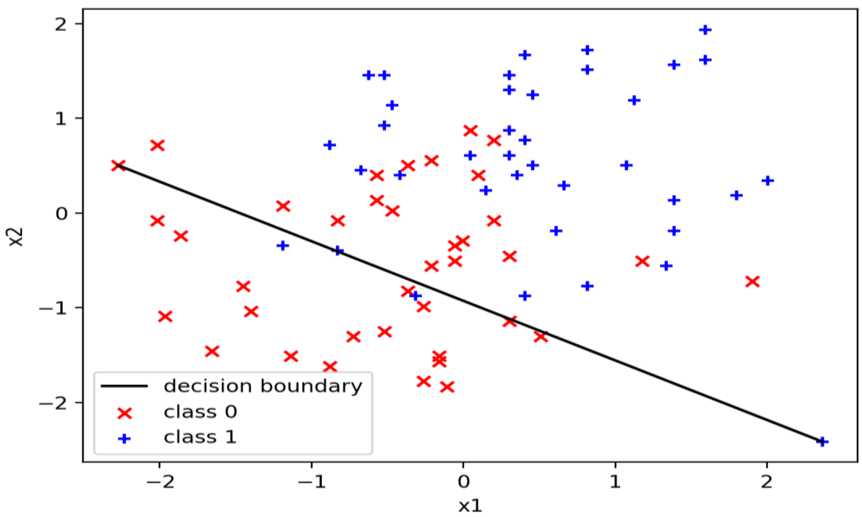

decision boundary
class 0
class 1
x1
x2
−2
−1
0
1
2

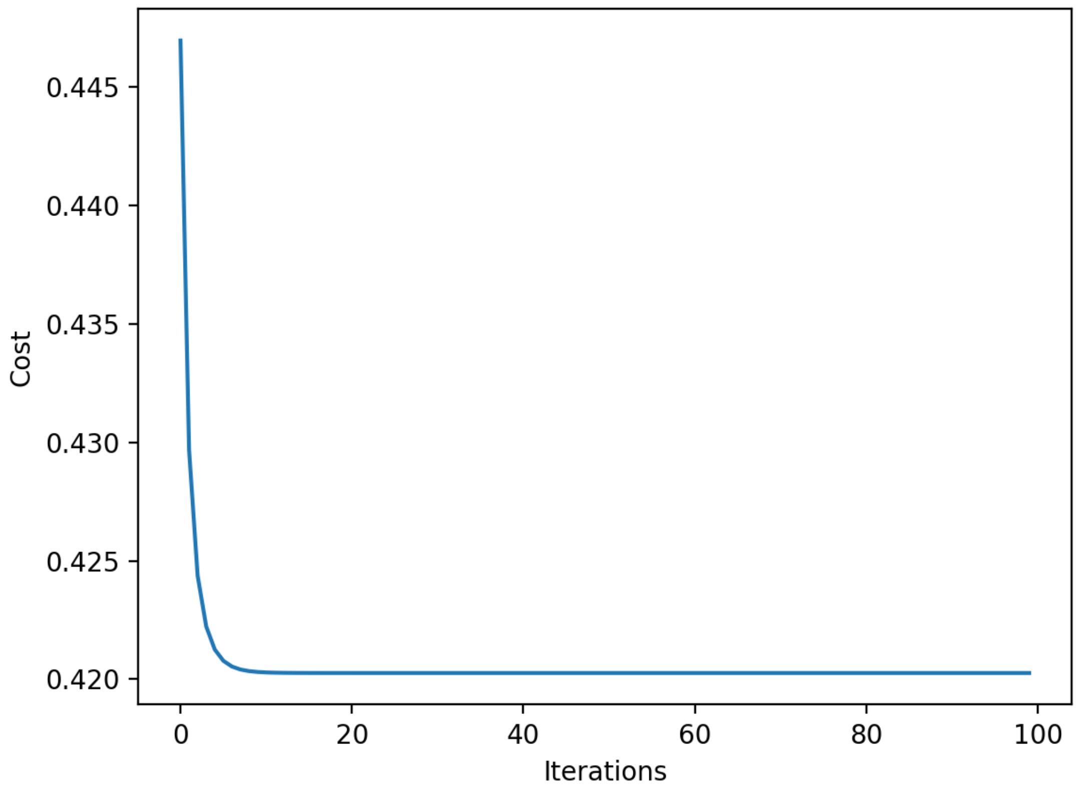

Cost
Iterations
0.42
0.41
0.40
0.39
0.38
0.37
0
20
40
60
80
100

For sample 77

Final training cost: 0.42026
Minimum training cost: 0.42026, on iteration #60
Final test cost: 0.02577




decision boundary
class 0
class 1
x1
x2

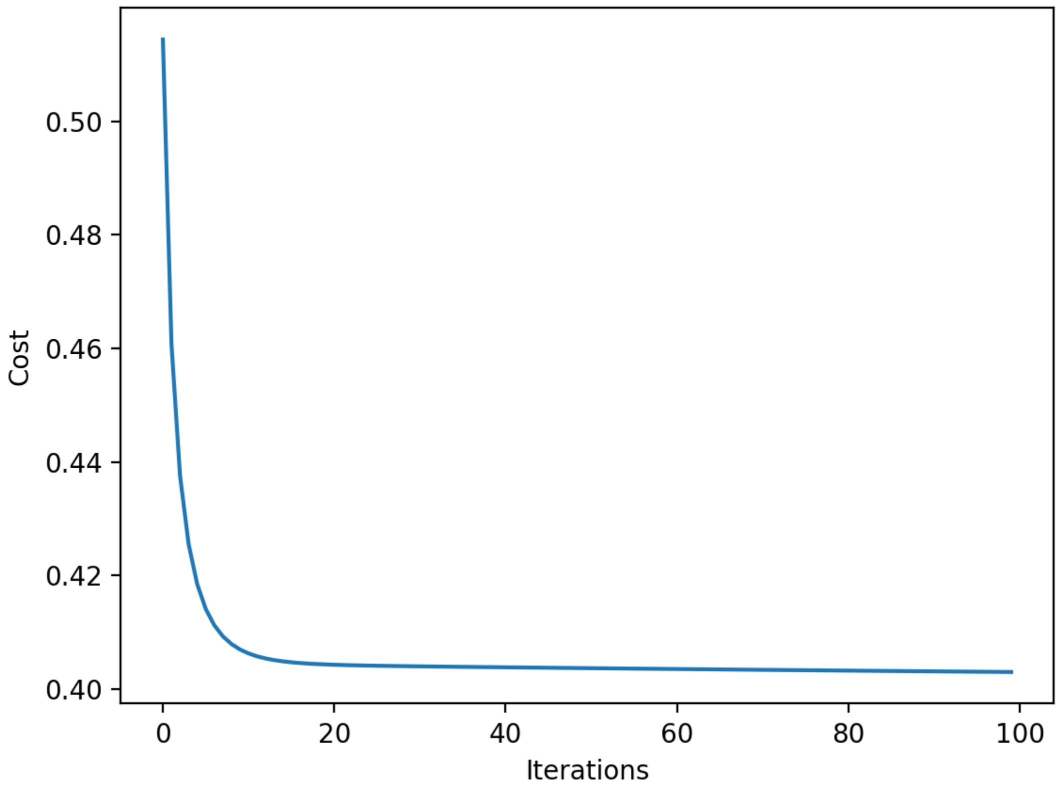

Cost
Iterations

The training set must be separate from the test set whereas the training phase consumes the training set. In order to find a set of parameter values that minimise a certain cost function over the whole training set.
In the case of test set it is for testing the model to check how it performs on new unseen versions of the same classes.

```
x1 = X[..., 0]
x2 = X[..., 1]
X = np.column_stack((X, x1*x2))
X = np.column_stack((X, x1**2))
X = np.column_stack((X, x2**2))
```

```
alpha = 0.01

iterations = 100
```

```
theta = np.zeros((6))
```



In task 8 I am modifying the function gradient_descent_training.py to store the current cost for training set and testing set.
I am storing the cost of the training set to cost_vector_train and for the test set to cost_vector_test.

```
for j in range(len(theta))
```

```
hypothesis = calculate_hypothesis(X,
theta_temp, i)
```

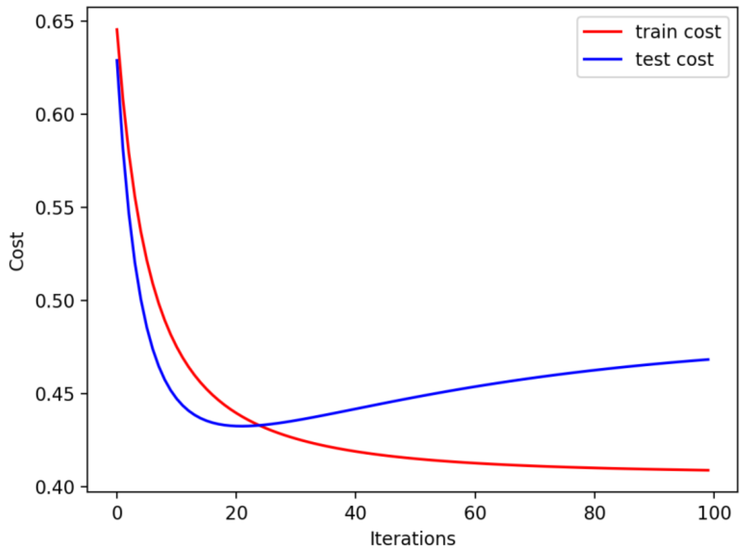
```
#######################################/
sigma[j] += (hypothesis - output) * X[i, j]
```

Here I have updated the themp

```
theta_temp[j] = theta_temp[j] - alpha * sigma[j]
```

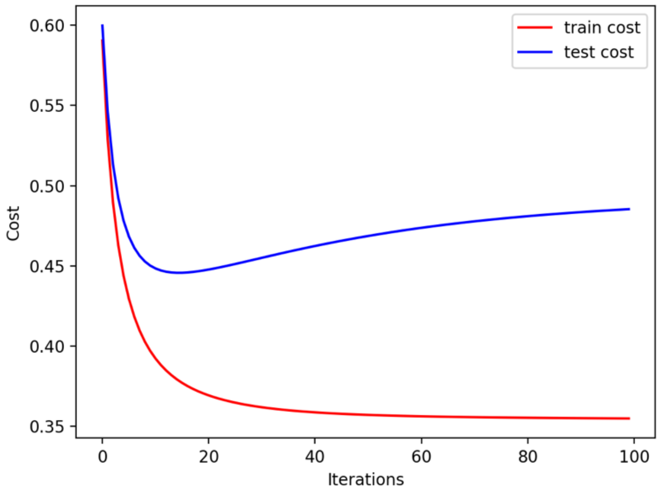
after running the ml_assgn_4.py file I can see the following observations:

sample 20
Final train cost: 0.43985
Minimum train cost: 0.43985, on iteration #100
Final test cost: 0.40566
Minimum test cost: 0.40566, on iteration #100



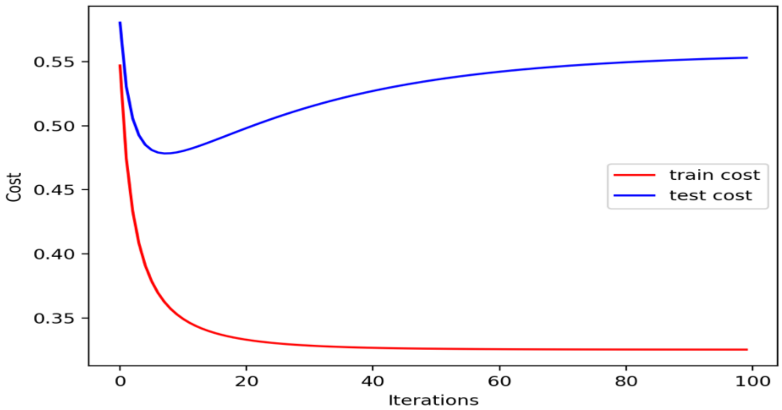
sample 30

Final train cost: 0.35294
Minimum train cost: 0.35294, on iteration #100
Final test cost: 0.49431
Minimum test cost: 0.45083, on iteration #16



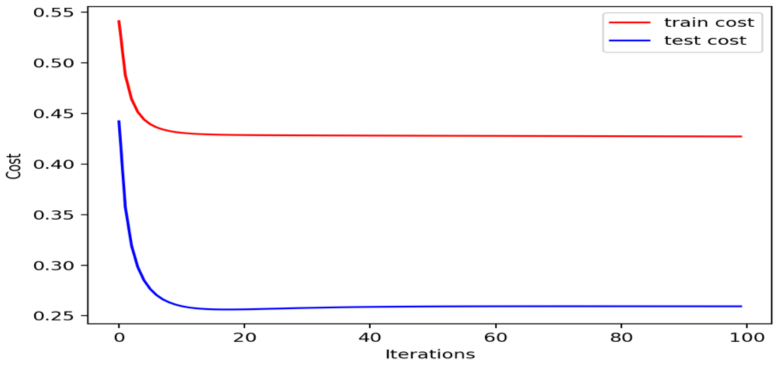
sample 40

Final train cost: 0.26615
Minimum train cost: 0.26615, on iteration #100
Final test cost: 0.65850
Minimum test cost: 0.51458, on iteration #7



sample 70

Final train cost: 0.40522
Minimum train cost: 0.40522, on iteration #100
Final test cost: 0.40428
Minimum test cost: 0.39321, on iteration #7



In assgn1_ex5.py I am adding extra features and analysing the effect.

```
def gradient_descent_training(X_train, y_train,
X_test, y_test, theta, alpha, iterations):
```

```
output1 = y_train[i]
hypothesis = calculate_hypothesis(X_train,
theta_temp, i)
```

```

```

```
for j in range(len(theta)):
```

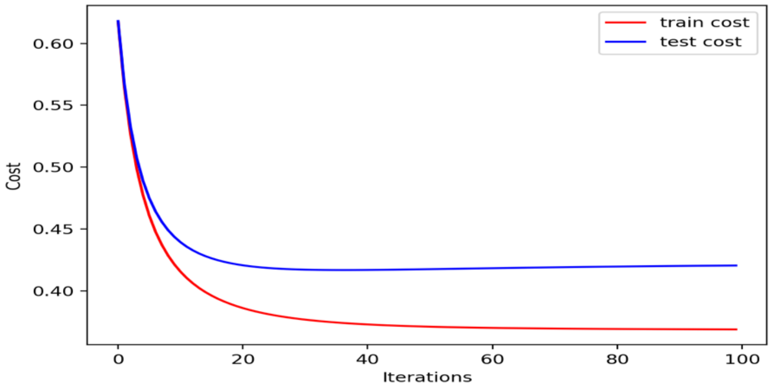
```
output1 = y_train[i]
hypothesis = calculate_hypothesis(X_train,
theta_temp, i)
```

```
sigma[j] += (hypothesis - output1) * X_train[i,
j]
```

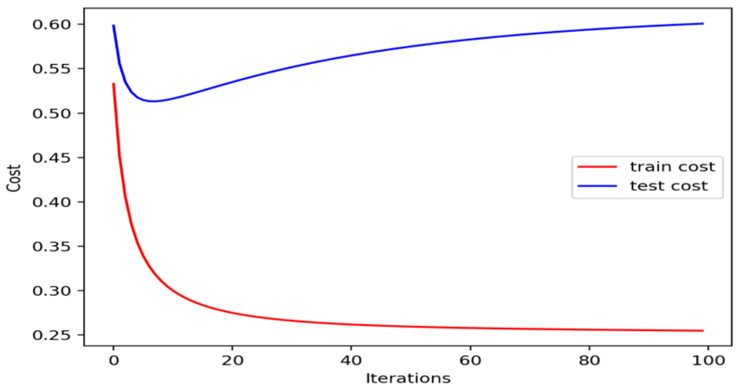
```
it_cost_vector_train = compute_cost(X_train,
y_train, theta)
it_cost_vector_test = compute_cost(X_test,
y_test, theta)

cost_vector_train =
np.append(cost_vector_train,
it_cost_vector_train)
cost_vector_test = np.append(cost_vector_test,
it_cost_vector_test)
```

Sasmple 20

Final train cost: 0.36887
Minimum train cost: 0.36887, on iteration #100
Final test cost: 0.42051
Minimum test cost: 0.41684, on iteration #37

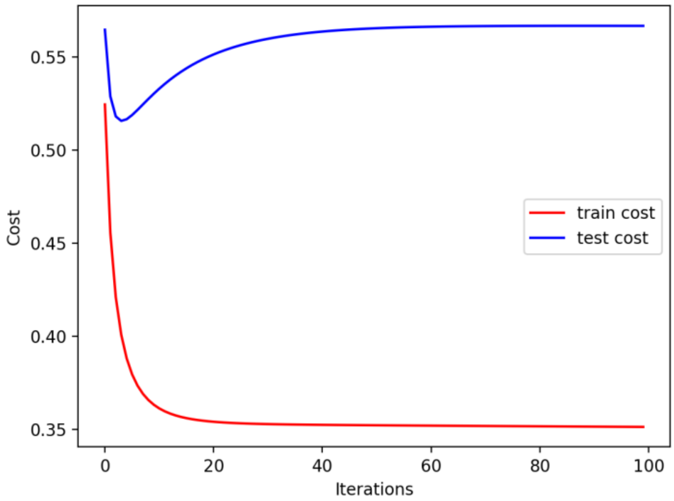


Sample 40



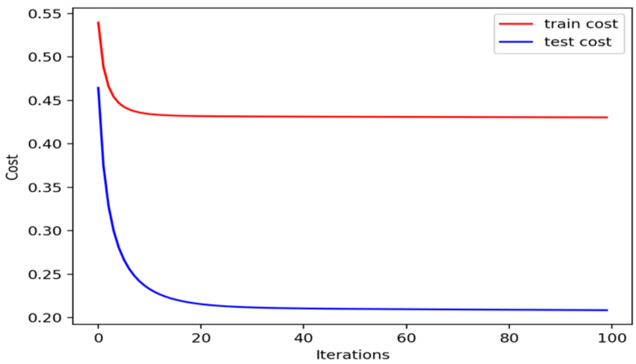
Final train cost: 0.42920
Minimum train cost: 0.42920, on iteration #100
Final test cost: 0.41132
Minimum test cost: 0.40567, on iteration #19

Sample 60



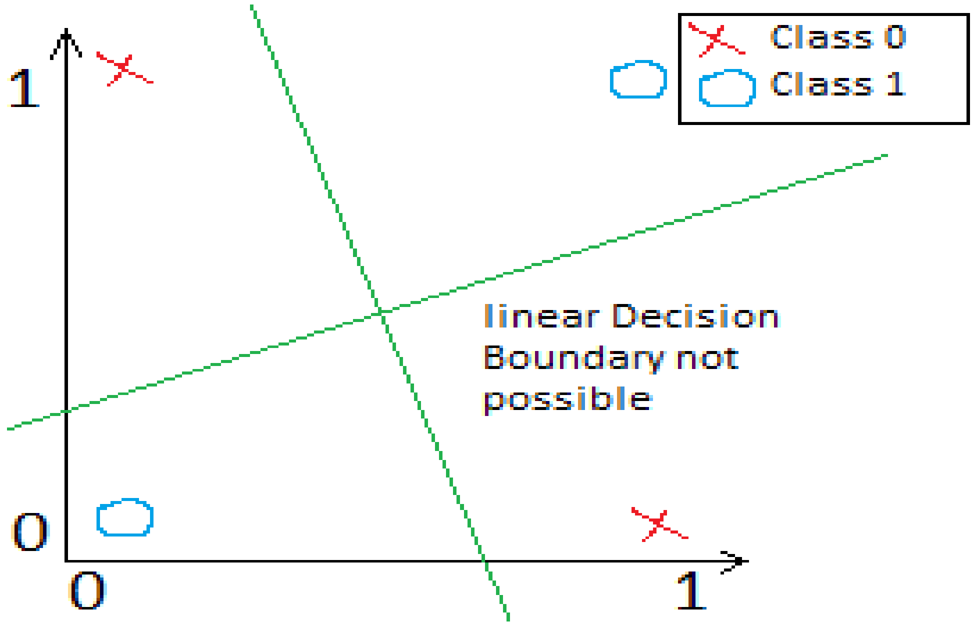
Final train cost: 0.39530
Minimum train cost: 0.39530, on iteration #100
Final test cost: 0.44090
Minimum test cost: 0.42626, on iteration #10

Sample 70



Final train cost: 0.43500
Minimum train cost: 0.43500, on iteration #100
Final test cost: 0.17417
Minimum test cost: 0.17385, on iteration #56

The logistic regression unit cannot solve the XOR clasiification problem because the classes in XOR are not linearly seperable.



2. Neural Network

In my first case I am modifying sigmoid.py to use the sigmoid function.

```
output = 1.0 / (1.0 + np.exp(-z))

return output
```

```
def backward_pass(self, inputs, targets,
learning_rate):
```

Step 1: In step 1 the output deltas are used to update the weights of the output layer.

```
output_deltas = (outputs - targets) *
sigmoid_derivative(sigmoid(outputs[i]))
```

Step 2: Hidden deltas are used to update the weights of the hidden layer.

Note: Here I need to backpropagate the error to the hidden neurons

```
hidden_deltas[i] =
sigmoid_derivative(sigmoid(self.y_hidden[i]))*sigma
```

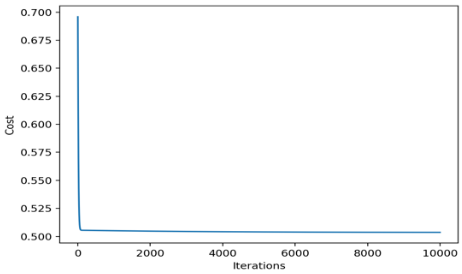
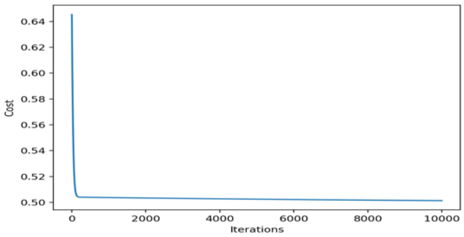
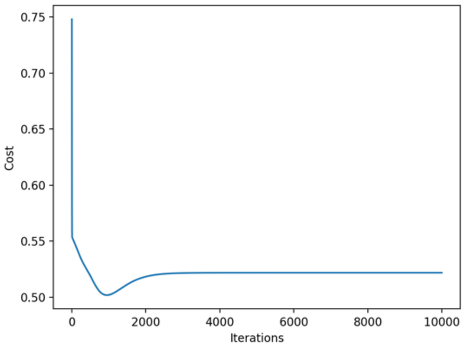
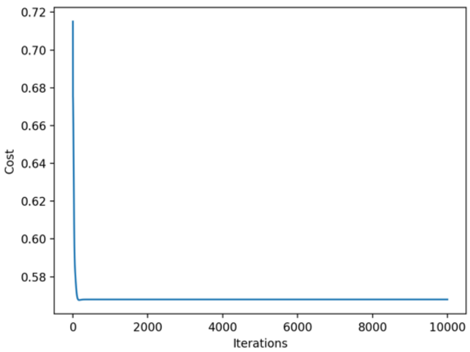
Step 3: Her I am updating the output weights, the connections from the hidden neurons to the output neurons.

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

Step4: Here I am updating the hidden weights, the connections from the hidden neurons to the inputs.

```
self.w_hidden[i, j] = self.w_hidden[i, j] -
learning_rate * hidden_deltas[j] * sigmoid(inputs[i])
```

Learning rate	Cost Function

0.1	Cost Iterations 0.700 0.675 0.650 0.625 0.600 0.575 0.550 0.525 0.500 0 2000 4000 6000 8000 10000
0.05	Cost Iterations 0.64 0.62 0.60 0.58 0.56 0.54 0.52 0.50 0 2000 4000 6000 8000 10000
1.0	Cost Iterations 0.75 0.70 0.65 0.60 0.55 0.50 0 2000 4000 6000 8000 10000
3	Cost Iterations 0.72 0.70 0.68 0.66 0.64 0.62 0.60 0.58 0 2000 4000 6000 8000 10000

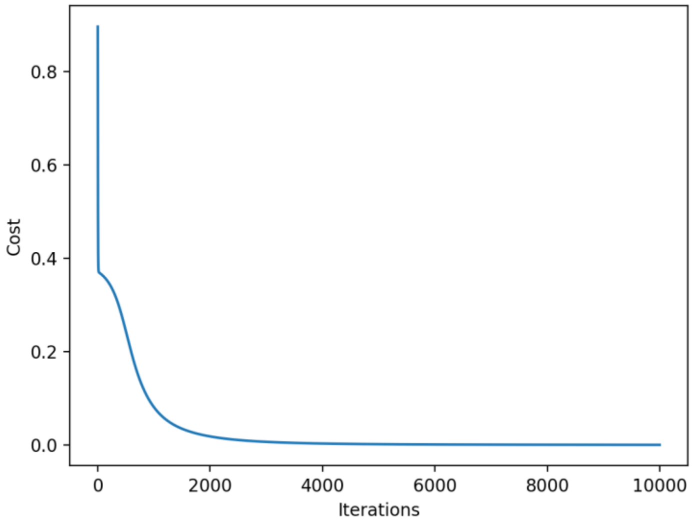
In task 2 I am checking for the NOR gate and with the truth table given below:

2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

```
y = np.array([1, 0, 0, 0])
```

For the NOR gate output values I am getting the following samples of target and predicted value.

Sample #01 | Target value: 1.00 | Predicted value: 0.97625
Sample #02 | Target value: 0.00 | Predicted value: 0.00889
Sample #03 | Target value: 0.00 | Predicted value: 0.00905
Sample #04 | Target value: 0.00 | Predicted value: 0.00002
Minimum cost: 0.00037, on iteration #10000



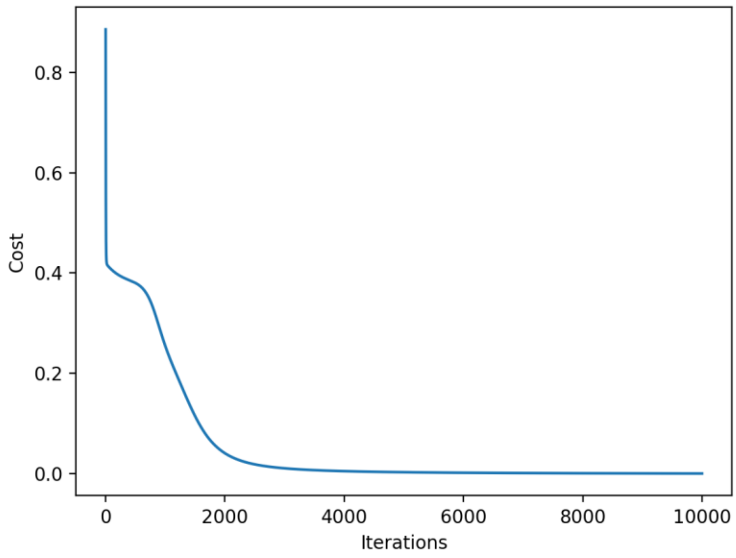
Now for AND gate with the truth table given below:

2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

```
y = np.array([0, 0, 0, 1])
```

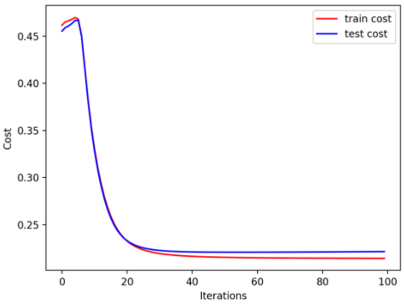
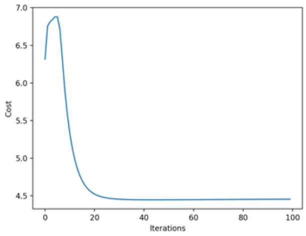
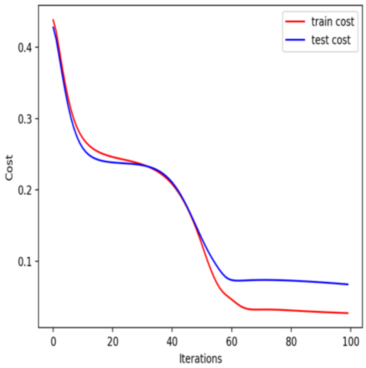
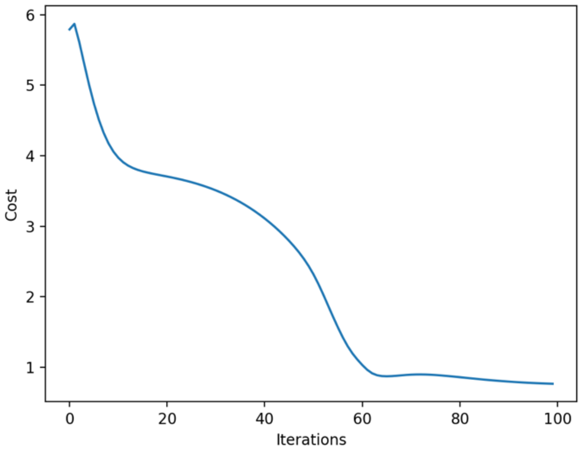
For the AND gate output values I am getting the following samples of target and predicted value.

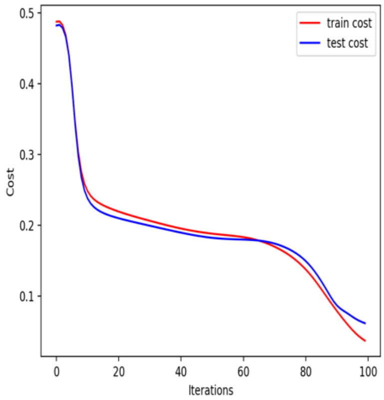
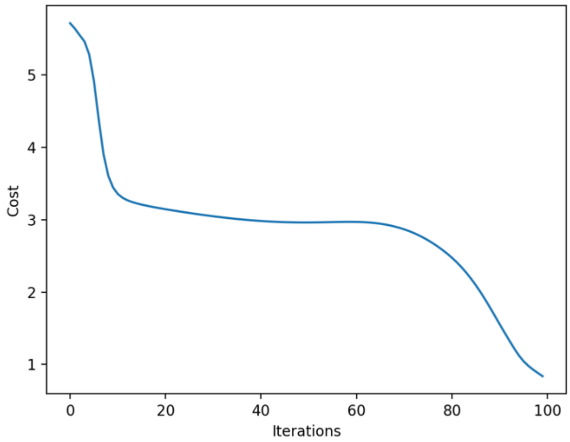
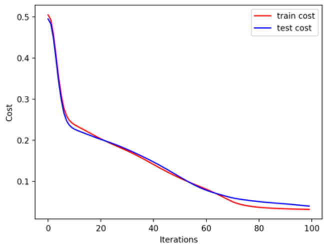
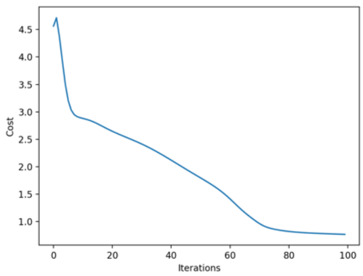
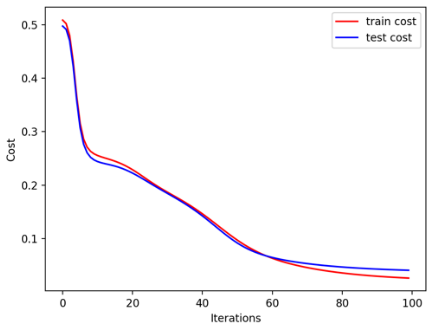
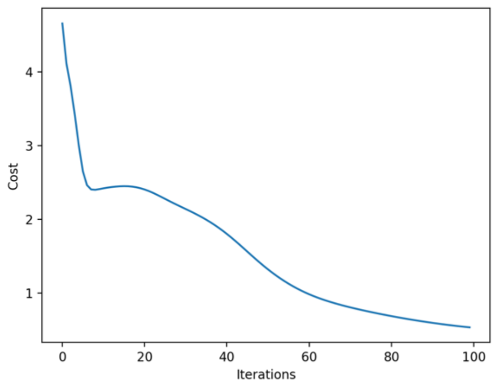
Sample #01 | Target value: 0.00 | Predicted value: 0.00032
Sample #02 | Target value: 0.00 | Predicted value: 0.01463
Sample #03 | Target value: 0.00 | Predicted value: 0.01463
Sample #04 | Target value: 1.00 | Predicted value: 0.96725
Minimum cost: 0.00085, on iteration #10000

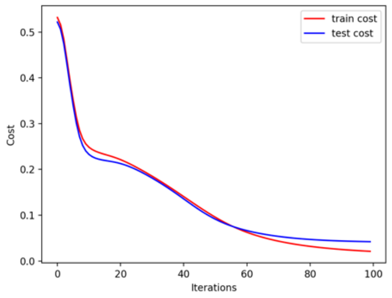
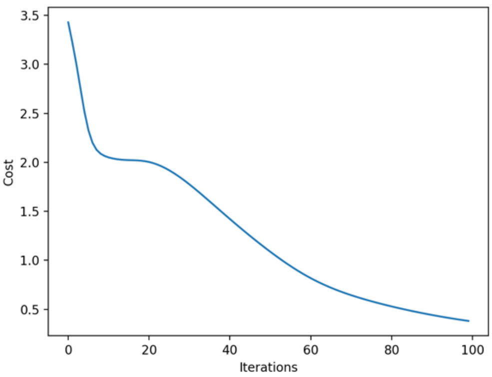


NOTE

- One versus all classification is a neat recipe to build a multiway classifier using just a series of binary classifiers.
- Use the one versus rest classification option makes it challenging to handle large datasets due to many class instances.

Hidden Neurons	Train Cost Test Code	Cost function
1		
2		
3		

<table>
<tr>
<td></td>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>5</td>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>7</td>
<td>
</td>
<td>
</td>
</tr>
</table>

10		

For hidden neuron 10 is the best for iteration.
We haven't generalized so we don't need these many basic neurons and it is fitting my training set well but it is not generalized.