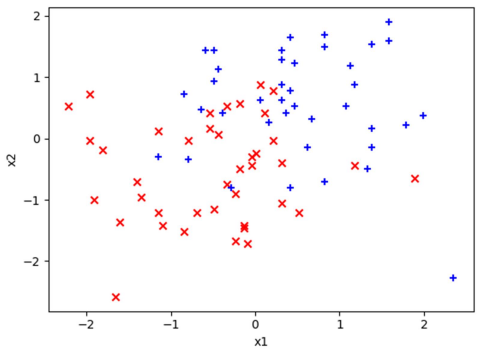
1. Logistic Regression

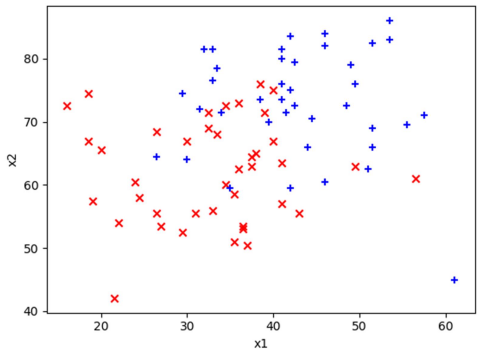
Task 1

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

Task 2

Normalized vs Not normalized





Task 3

```
########################################
# Write your code here
# You must calculate the hypothesis for the i-th sample of X, given X, theta and
i.
hypothesis = X[i].dot(theta)
########################################/
```

Task 4

```
########################################
# Write your code here
# You must calculate the cost
cost = -output * np.log(hypothesis) - (1 - output) * np.log(1 - hypothesis)
########################################/
```

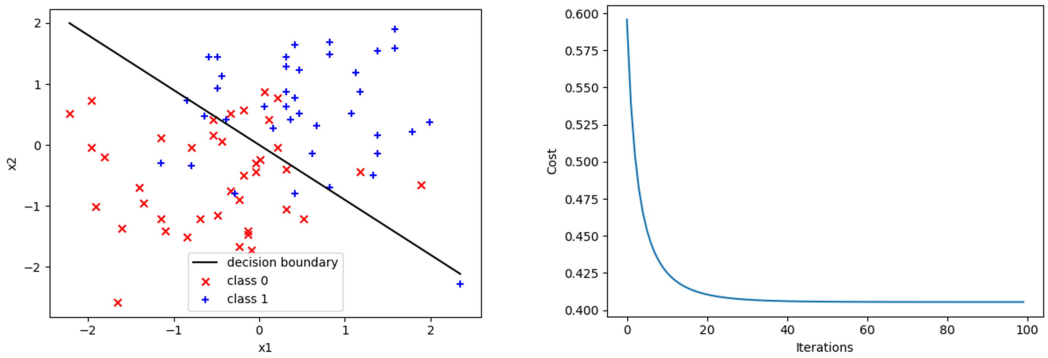
Task 5

```
########################################
# 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
min_x1 = np.amin(X,0)[1]
max_x1 = np.amax(X,0)[1]
```

```
x2_on_min_x1 = - (theta[0]*min_x1 + theta[1]*min_x1)/theta[2]
x2_on_max_x1 = - (theta[0]*max_x1 + theta[1]*max_x1)/theta[2]
##########################################/
```

The graph below was computed at

```
alpha = 0.8
iterations = 100
```



Task 6:

Sample 1

Final training cost: 0.20219

Minimum training cost: 0.20219, on iteration #100

Final test cost: 0.55341

Sample 2

Final training cost: 0.10128

Minimum training cost: 0.10128, on iteration #100

Final test cost: 0.82037

Sample 3

Final training cost: 0.32253

Minimum training cost: 0.32253, on iteration #100

Final test cost: 0.46167

Test cost is generally higher than training cost.

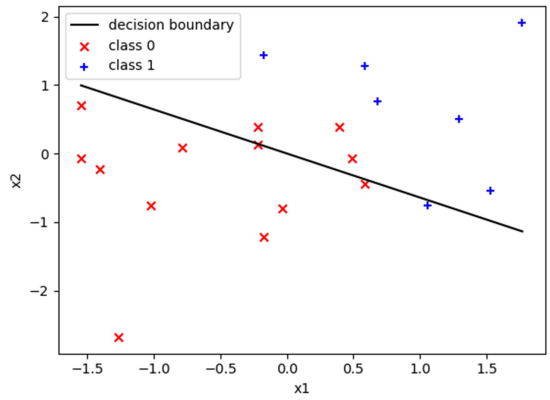
A bad generalization example given below:

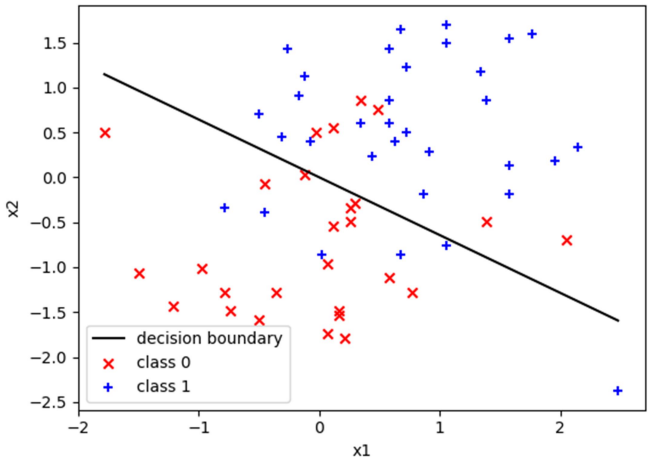
Final training cost: 0.16875

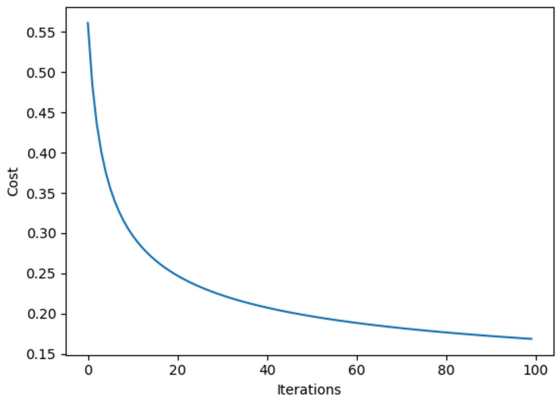
Minimum training cost: 0.16875, on iteration #100

Final test cost: 0.62992

Train, test and cost graphs are given below for the above stats







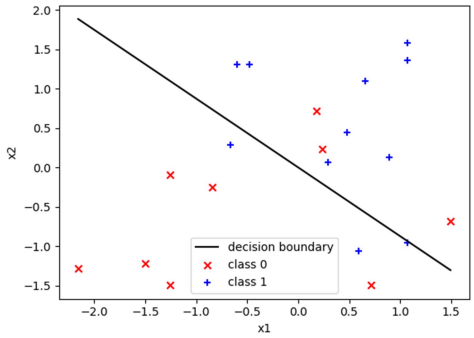
A good generalization example given below:

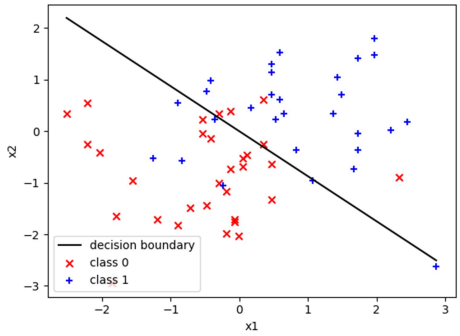
Final training cost: 0.44333

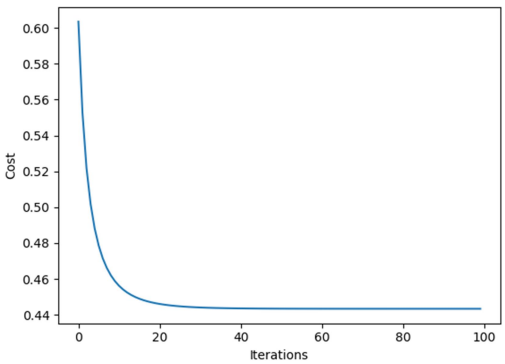
Minimum training cost: 0.44333, on iteration #100

Final test cost: 0.39462

Train, test and cost graphs are given below for the above stats







The training set generalizes well when it is able to provide an overview of the dataset. Therefore, due to random shuffling, the training dataset created can either result in good or bad models.

Task 7 :

```
# Create the features x1*x2, x1^2 and x2^2
########################################
# Write your code here
# Compute the new features
# Insert extra singleton dimension, to obtain Nx1 shape
# Append columns of the new features to the dataset, to the dimension of columns
(i.e., 1)

x1 = X[:, 0, None]
x2 = X[:, 1, None]
X = np.append(X, x1*x2, axis=1)
X = np.append(X, x1*x1, axis=1)
X = np.append(X, x2*x2, axis=1)
########################################/
```

Final cost: 0.40261

Minimum cost: 0.40261, on iteration #100

The training error is generally less. This is due to overfitting as the number of samples is the same but the attributes have increased.

Task 8:

```
# Create the features x1*x2, x1^2 and x2^2
########################################
# Write your code here
# Compute the new features
# Insert extra singleton dimension, to obtain Nx1 shape
# Append columns of the new features to the dataset, to the dimension of columns
(i.e., 1)
x1 = X[:, 0, None]
x2 = X[:, 1, None]
X = np.append(X, x1*x2, axis=1)
X = np.append(X, x1*x1, axis=1)
X = np.append(X, x2*x2, axis=1)
########################################/
```

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

```
# 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_train = compute_cost(X_train, y_train, theta)
```

```
cost_vector_train = np.append(cost_vector_train, iteration_cost_train)
iteration_cost_test = compute_cost(X_test, y_test, theta)
cost_vector_test = np.append(cost_vector_test, iteration_cost_test)
########################################/
```

Sample 1

Final train cost: 0.39157

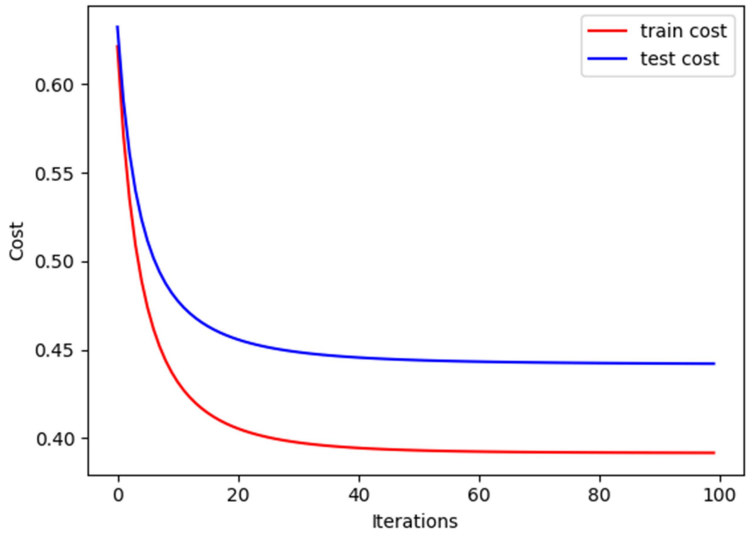
Minimum train cost: 0.39157, on iteration #100

Final test cost: 0.44199

Minimum test cost: 0.44199, on iteration #100

Split 60-20 between training and testing

The graph is given below:



A split of 60-20 seems ideal. Few of the examples with various other splits are given below.

Sample 2

Final train cost: 0.41329

Minimum train cost: 0.41329, on iteration #100

Final test cost: 0.34544

Minimum test cost: 0.34544, on iteration #100

Here testing error is less but training error is more

Sample 3

Final train cost: 0.38031

Minimum train cost: 0.38031, on iteration #100

Final test cost: 0.45982

Minimum test cost: 0.42988, on iteration #21

Sample 4

Final train cost: 0.10920

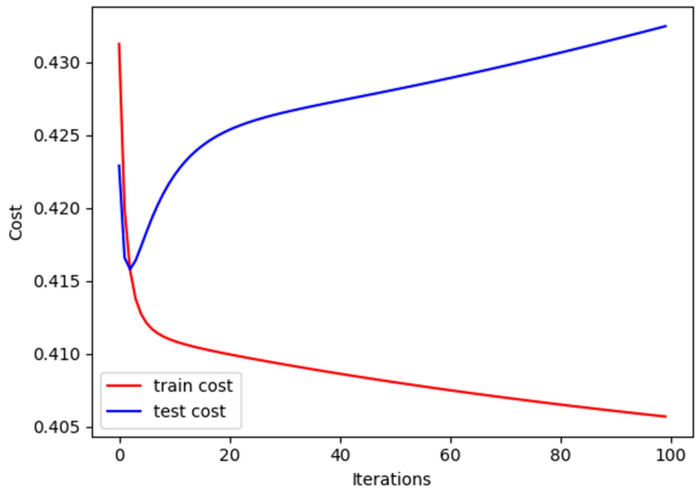
Minimum train cost: 0.10920, on iteration #100

Final test cost: 0.53349

Minimum test cost: 0.43937, on iteration #16

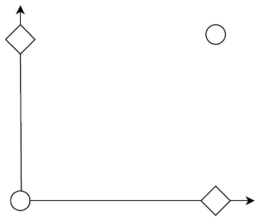
```
# Create the features x1*x2, x1^2, x2^2, x1^3 and x2^3
##########################################
# Write your code here
# Compute the new features
# Insert extra singleton dimension, to obtain Nx1 shape
# Append columns of the new features to the dataset, to the dimension of columns
(i.e., 1)
x1 = X[:, 0, None]
x2 = X[:, 1, None]
X = np.append(X, x1*x2, axis=1)
X = np.append(X, x1*x1, axis=1)
X = np.append(X, x2*x2, axis=1)
X = np.append(X, x1*x1*x1, axis=1)
X = np.append(X, x2*x2*x2, axis=1)
##########################################/
```

For training set at 70 and test at 10, the cost for test is still lower. With a decrease in the training samples, the test cost increases. As one can see in the graph below, the model has started overfitting the training set as the number of instances is not enough to avoid overfiiting. The graph was plotted at training sample =50.



Task 9:

Logistic regression can't solve XOR problem as it can't create complicated decision boundaries.



A linear classifier cannot create a boundary such that it can differentiate both the classes accurately. For a complicated boundary, usually neural nets are preferred. As the output of the first layer serves as the input to the next layer, hence a more complicated boundary is generated.

Task 10

```
output_deltas = (outputs - targets) * sigmoid_derivative(outputs)
# for i in range(self.n_out):
    ########################################
    # Write your code here
    # compute output_deltas : delta_k = (y_k - t_k) * g'(x_k)
    # the code below doesn't work for xor Example as the output is a scalar
    # output_deltas[i] = (outputs[i] - targets[i]) *
sigmoid_derivative(outputs[i])
    ########################################/
```

```
# 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_weight_delta = 0
    for j in range(len(output_deltas)):
        sum_weight_delta = sum_weight_delta + self.w_out[i, j] * output_deltas[j]
    hidden_deltas[i] = sigmoid_derivative(self.y_hidden[i]) * sum_weight_delta

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

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

```
# 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] -
learning_rate*hidden_deltas[j]*inputs[i]
        ########################################/
```

Xor at local minima

Sample #01 | Target value: 0.00 | Predicted value: 0.46942

Sample #02 | Target value: 1.00 | Predicted value: 0.50835

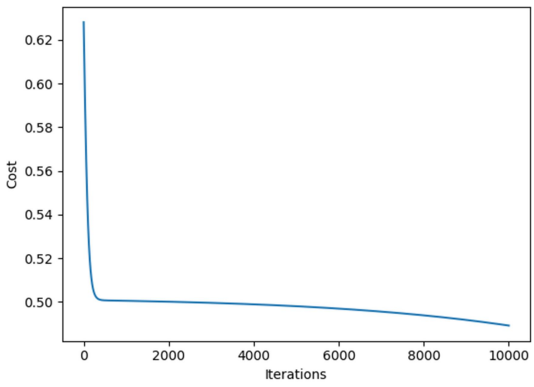
Sample #03 | Target value: 1.00 | Predicted value: 0.50929

Sample #04 | Target value: 0.00 | Predicted value: 0.52256

Minimum cost: 0.48910, on iteration #10000

The graph below was computed by the stats

```
n_hidden = 2
iterations = 10000
learning_rate = 0.2
```



A better solution for Xor

Sample #01 | Target value: 0.00 | Predicted value: 0.02701

Sample #02 | Target value: 1.00 | Predicted value: 0.96197

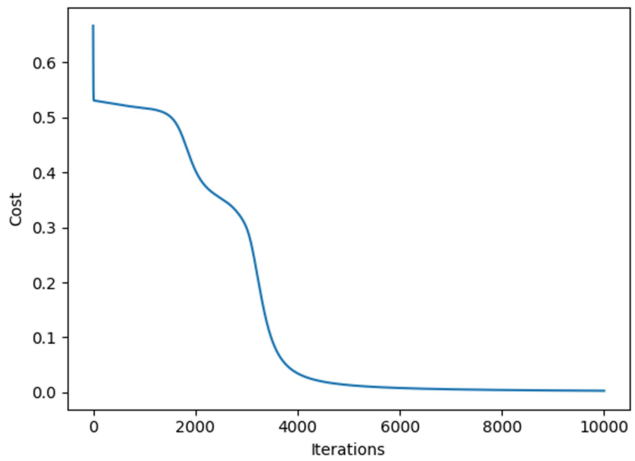
Sample #03 | Target value: 1.00 | Predicted value: 0.96182

Sample #04 | Target value: 0.00 | Predicted value: 0.04925

Minimum cost: 0.00303, on iteration #10000

The graph below was computed by the stats

```
n_hidden = 2
iterations = 10000
learning_rate = 0.5
```



Task 11

Other gates implemented are NAND

Sample #01 | Target value: 1.00 | Predicted value: 0.99220

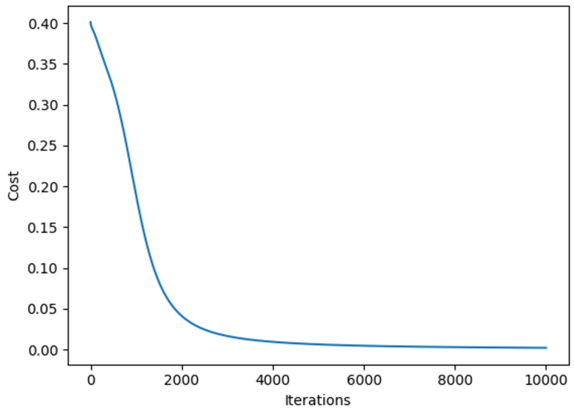
Sample #02 | Target value: 1.00 | Predicted value: 0.96735

Sample #03 | Target value: 1.00 | Predicted value: 0.96752

Sample #04 | Target value: 0.00 | Predicted value: 0.04790

Minimum cost: 0.00224, on iteration #10000

```
n_hidden = 2
iterations = 10000
learning_rate = 0.2
```



Other gates implemented are NOR

Sample #01 | Target value: 1.00 | Predicted value: 0.95819

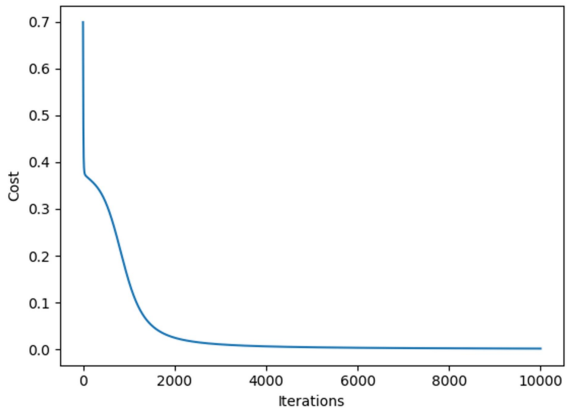
Sample #02 | Target value: 0.00 | Predicted value: 0.02751

Sample #03 | Target value: 0.00 | Predicted value: 0.02752

Sample #04 | Target value: 0.00 | Predicted value: 0.01340

Minimum cost: 0.00172, on iteration #10000

```
n_hidden = 2
iterations = 10000
learning_rate = 0.2
```



Task 12:

The Iris data set contains three different classes of data that we need to discriminate between. The classification can be achieved by using multinomial logistic regression (multi class classification). Multinominal logistic regression uses the concept of basic logistic regression but generalizes it for multiple classes.

Whereas, the same problem can be solved using neural networks which has different layers of neurons that help in classification.

Task 13:

Hidden_neurons = 1

Minimum cost: 3.07459, on iteration #24

Hidden_neurons = 2

Minimum cost: 0.10176, on iteration #100

Hidden_neurons = 3

Minimum cost: 0.05161, on iteration #100

Hidden_neurons = 5

Minimum cost: 0.03815, on iteration #100

Hidden_neurons = 7

Minimum cost: 0.03430, on iteration #100

Hidden_neurons = 10

Minimum cost: 0.03126, on iteration #100

The train cost of the neural net is going down with the increase in the number of hidden neurons. But the test cost starts to go up after hidden neurons =2. Thus the best number would be 1. The alpha is set at 2.3 and the number of iterations is set at 100.