ECS708P Machine Learning: Assignment 1 Part 2

Animesh Devendra Chourey

210765551

1. Logistic Regression

Logistic Regression is a supervised learning technique. Here we predict results in a discrete outpy, meaning we are trying to map input variables into discrete categories. We try to fit our dataset with the help of gradient descent algorithm which basically helps us find the best parameters.

Task 1:

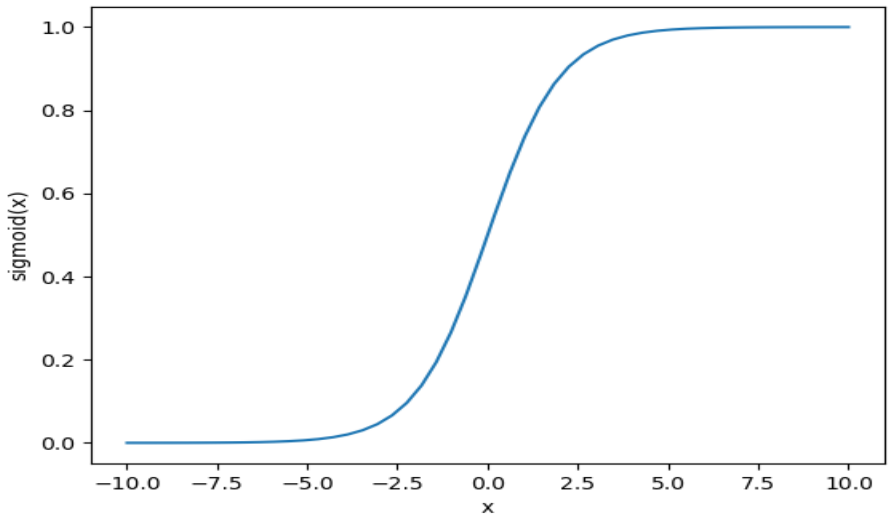
plot_sigmoid.py

plot_sigmoid() (To use plot_sigmoid function we need to call it first in plot_sigmoid.py)

plt.show() (To plot the sigmoid function we need to add this line to plot_sigmoid function)

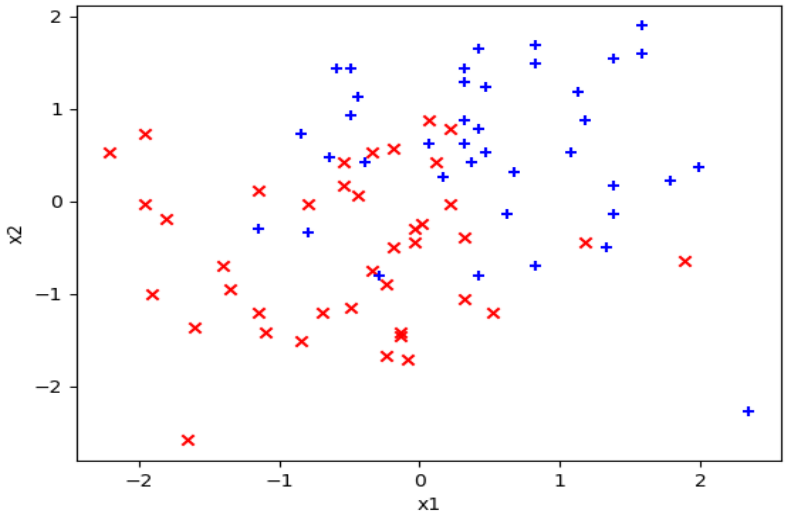
sigmoid.py

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

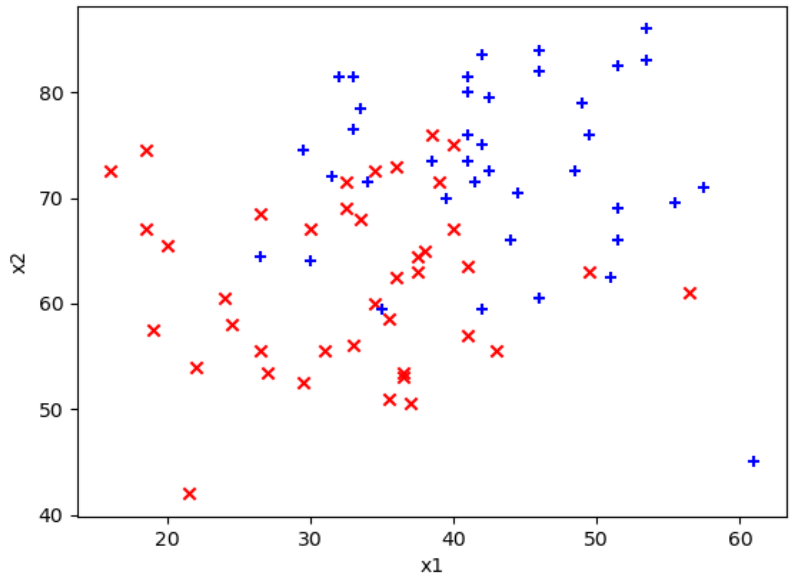


Task 2:

Normalized Data



Not Normalized Data



1.1 Cost function and gradient for logistic regression

Task 3

calculate_hypothesis.py

```
for j in range(len(theta)):
    hypothesis = hypothesis + theta[j] * X[i][j]
```

Task 4

compute_cost.py

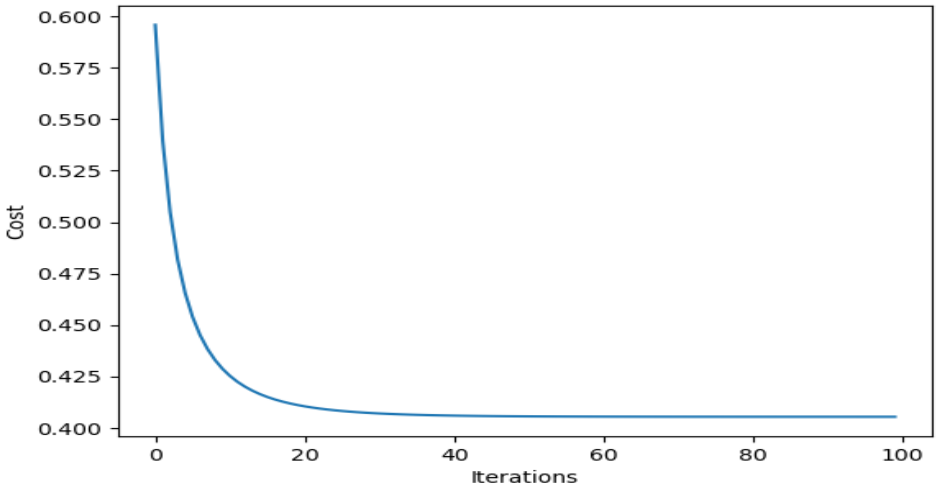
```
cost = - (output * np.log(hypothesis) ) - ( (1-output) * np.log(1-hypothesis) )
```

gradient_descent.py

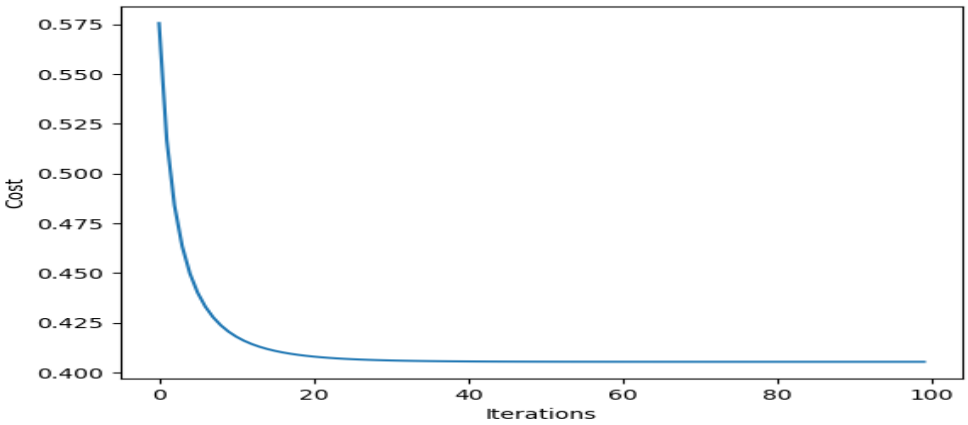
```
hypothesis = calculate_hypothesis(X, theta, i)

sigma = sigma + (hypothesis - output) * X[i]

theta_temp = theta_temp - sigma * (alpha/m)
```

Alpha	Minimum Cost
1.0	0.40545
0.8	0.40545
3	0.40545

Minimum Cost does not change for multiple values of α.



α = 0.8



α = 1.0

1.2 Draw the decision boundary

Task 5

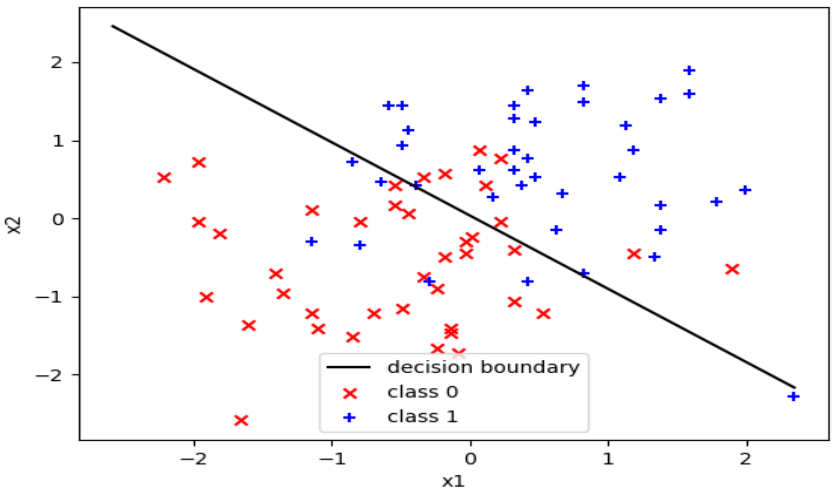
plot_boundary.py

min_x1 = np.min(X)

max_x1 = np.max(X)

x2_on_max_x1 = (-theta[0] - theta[1] * max_x1) / theta[2]

x2_on_min_x1 = (-theta[0] - theta[1] * min_x1) / theta[2]



1.3 Non-linear features and overfitting

Task 6

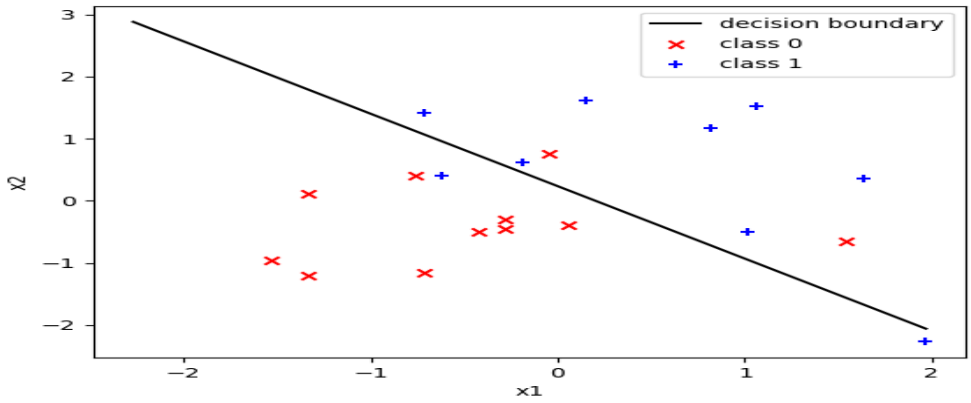
Execution	Final Training Cost	Minimum Training Cost	Final Test Cost
Execution 1	0.44590	0.44590	0.43390
Execution 2	0.41535	0.41535	0.41123
Execution 3	0.21637	0.41123	0.59711

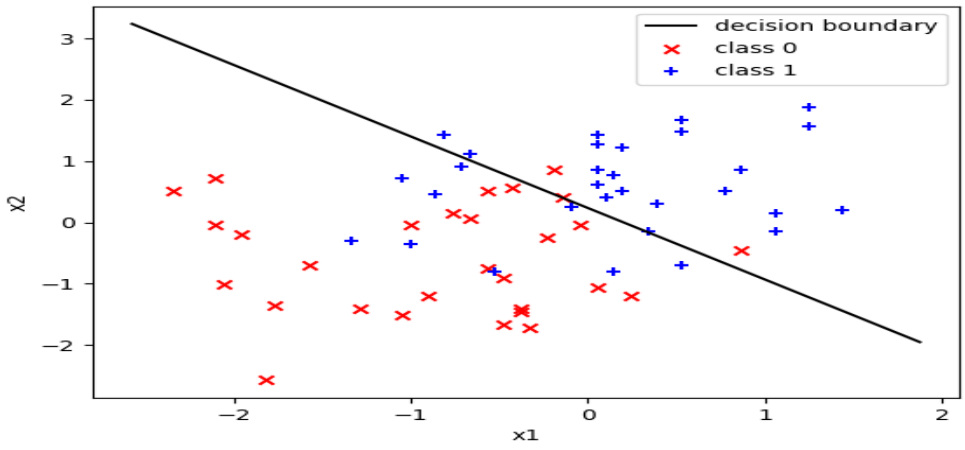
The training set generalize well when it is performing well enough on the training dataset and as well as on the test dataset. Otherwise either it will underfit i.e. it will not perform well on training data or overfit i.e. it will perform well on training set but not on test set.

Training Cost : The training cost is calculated over the training dataset i.e. the same model the algorithm was trained on.

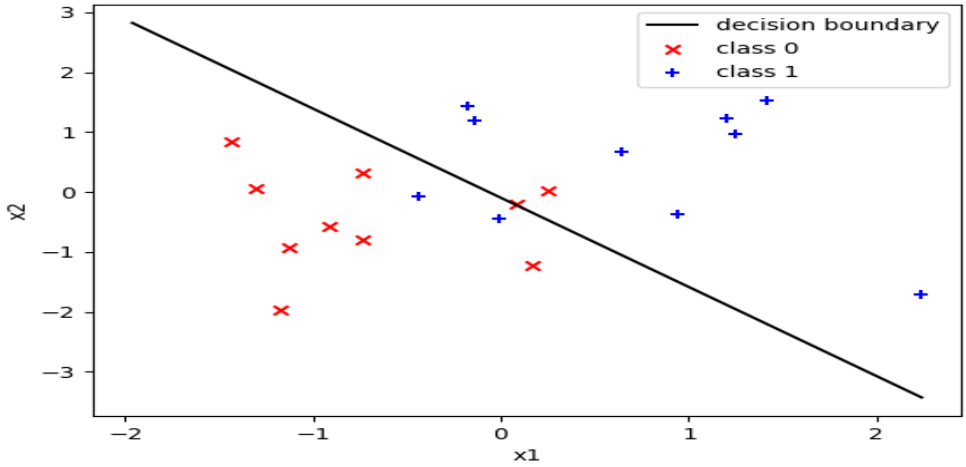
Test Cost : Test cost is how well our trained algorithm is predicting on the test dataset.

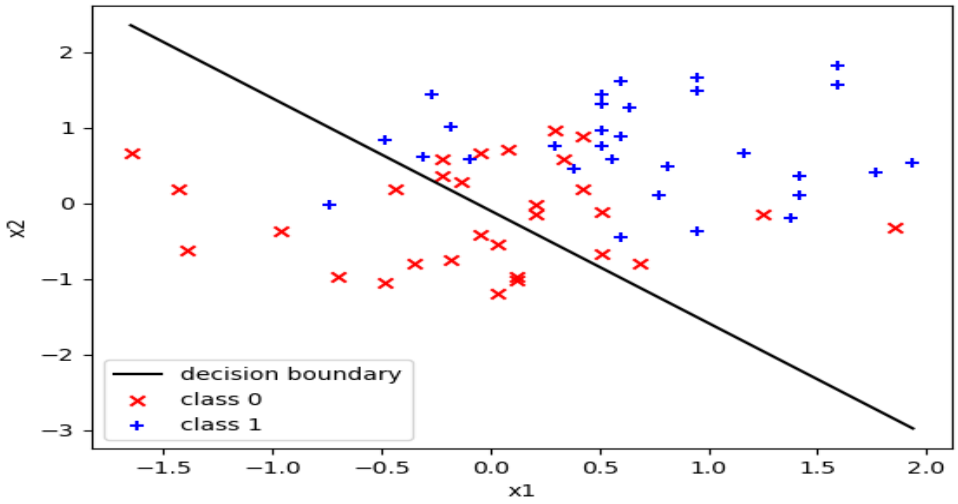
Good Split





Bad Split





ml_assgn1_ex3.py

Created the new features:

X3 = X[:,0] * X[:,1]

X4 = X[:,0] ** 2

X5 = X[:,1] ** 2

X3 = np.reshape(X3, (X.shape[0],1))

X4 = np.reshape(X4, (X.shape[0],1))

X5 = np.reshape(X5, (X.shape[0],1))

```
X = np.append(X, X3,axis=1)

X = np.append(X, X4,axis=1)

X = np.append(X, X5,axis=1)
```

Initialize parameters:

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

Task 7

Final cost: 0.40261

Minimum cost: 0.40261, on iteration #100

The cost seems to be going down which ultimately means that the error has reduced. One reason could be that because the features are increased and the equation becomes a quadratic equation the data might be able to fit just right to the classification algorithm.

Task 8

ml_assgn1_ex4.py

```
X3 = X[:,0] * X[:,1]

X4 = X[:,0] ** 2

X5 = X[:,1] ** 2


X3 = np.reshape(X3, (X.shape[0],1))

X4 = np.reshape(X4, (X.shape[0],1))

X5 = np.reshape(X5, (X.shape[0],1))


X = np.append(X, X3,axis=1)

X = np.append(X, X4,axis=1)

X = np.append(X, X5,axis=1)
```

Initialize parameters:

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

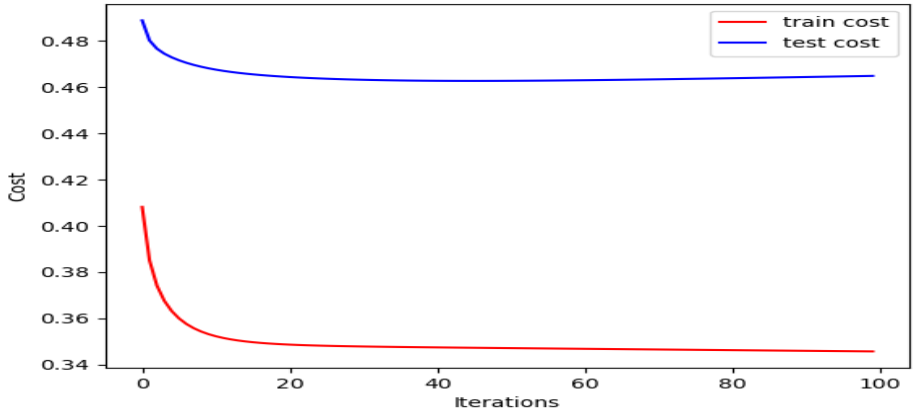
gradient_descent_training.py

```
hypothesis = calculate_hypothesis(X_train, theta, i)
sigma = sigma + (hypothesis - output) * X_train[i]
theta_temp = theta_temp - sigma * (alpha/m)
```

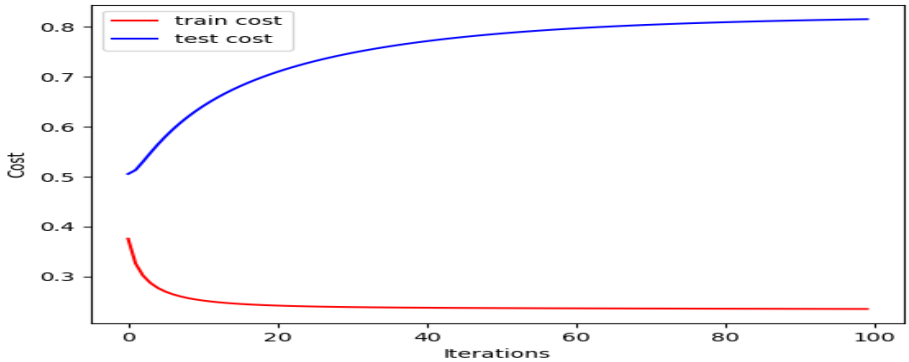
Storing the costs for both train and test set

```
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)
```

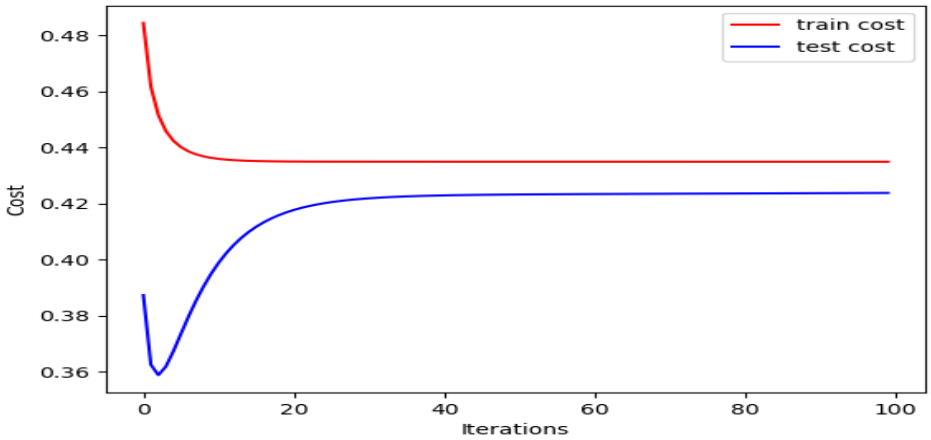
Alpha	Training Size	Final Train Cost	Minimum Train Cost	Final Test Cost	Minimum Test Cost
1.0	20	0.34570	0.34570 on iteration #100	0.46494	0.46277 on iteration #46
1.0	40	0.23507	0.23507 on iteration #100	0.81566	0.50547, on iteration #1
1.0	60	0.43497	0.43497, on iteration #100	0.42383	0.35885, on iteration #3



Train Size = 20



Train Size = 40



Train Size = 60

ml_assgn1_ex5.py

Created the new features:

```
X3 = X[:,0] * X[:,1]

X4 = X[:,0] ** 2

X5 = X[:,1] ** 2

X6 = X[:,0] ** 3

X7 = X[:,1] ** 3


X3 = np.reshape(X3, (X.shape[0],1))

X4 = np.reshape(X4, (X.shape[0],1))

X5 = np.reshape(X5, (X.shape[0],1))

X6 = np.reshape(X6, (X.shape[0],1))

X7 = np.reshape(X7, (X.shape[0],1))


X = np.append(X, X3,axis=1)

X = np.append(X, X4,axis=1)

X = np.append(X, X5,axis=1)

X = np.append(X, X6,axis=1)

X = np.append(X, X7,axis=1)
```

Initialize parameters:

theta = np.zeros((8))

Output:

Final train cost: 0.01851

Minimum train cost: 0.01851, on iteration #100

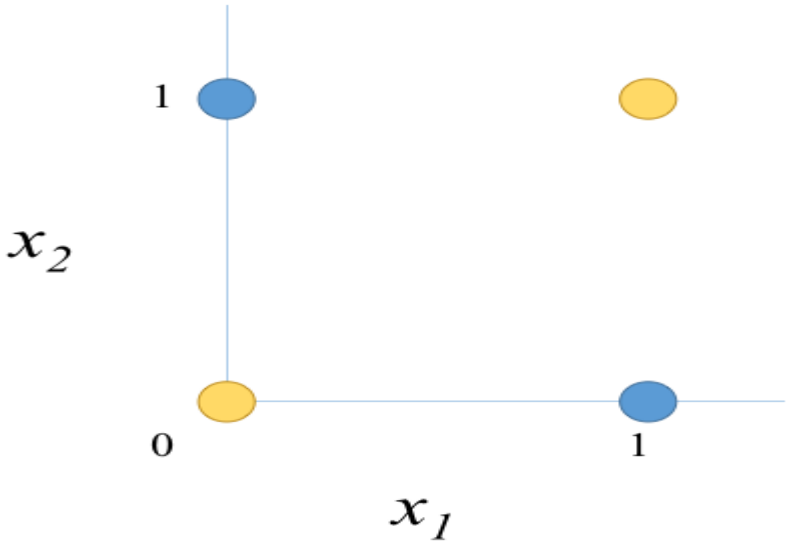
Final test cost: 1.14548

Minimum test cost: 0.48934, on iteration #1

Training cost has reduced drastically and the test cost has increased to an extreme level compared to before.
Making the equation of higher order polynomial, the hypothesis is able to fit the training data well. So basically it is overfitting our data and therefore the test cost is only going higher as the algorithm has only fit itself to train dataset and is not able to perform well on the test dataset.

Task 9

The Logistic Regression is not able to solve the XOR problem because the XOR outputs are not linearly seperable.



2. Neural Network

Neural Networks are the multi layered networks of neurons that we use to make predictions, to recognize paterns. Deep learning refers to training Neural Networks, sometimes extremely large neural networks.

Task 10

NeuralNetwork.py

Step -1

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

Step -2

```
sum_values = 0
 for j in range(len(output_deltas)):
                        sum_values = sum_values + self.w_out[i][j] * output_deltas[j]

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

Step-3

```
self.w_out[i][j] = self.w_out[i][j] - learning_rate * output_deltas[j] * self.y_hidden[i]
```

Step-4

```
self.w_hidden[i][j] = self.w_hidden[i][j] -  learning_rate * hidden_deltas[j] * inputs[i]
```

2.1 Implement backpropagation on XOR

Alpha	Minimum Cost
1.0	0.00028, on iteration #10000
0.1	0.01373, on iteration #10000
0.01	0.50055, on iteration #10000
0.2	0.00207, on iteration #10000
0.8	0.00038, on iteration #10000
→0.9	0.00190, on iteration #10000

For α = 0.9 we got the minimum cost.

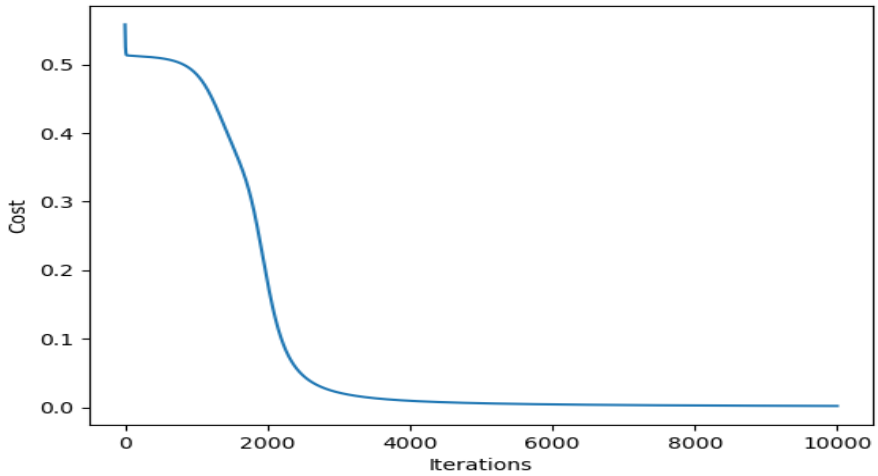


Fig α = 0.9

Task 11

xorExample.py

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

AND logical function applied " y = np.array([0, 0, 0, 1])"

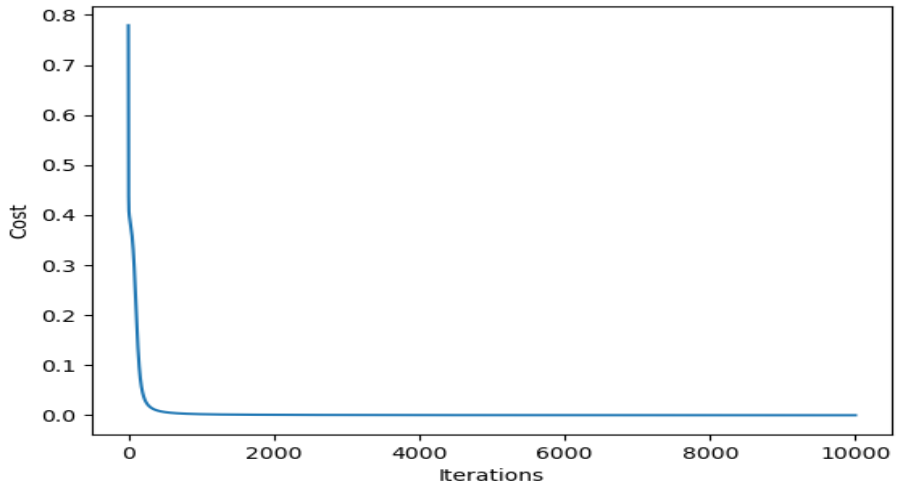


Fig α = 0.9 AND Logical Function

NOR logical function applied " y = np.array([1, 0, 0, 0])"

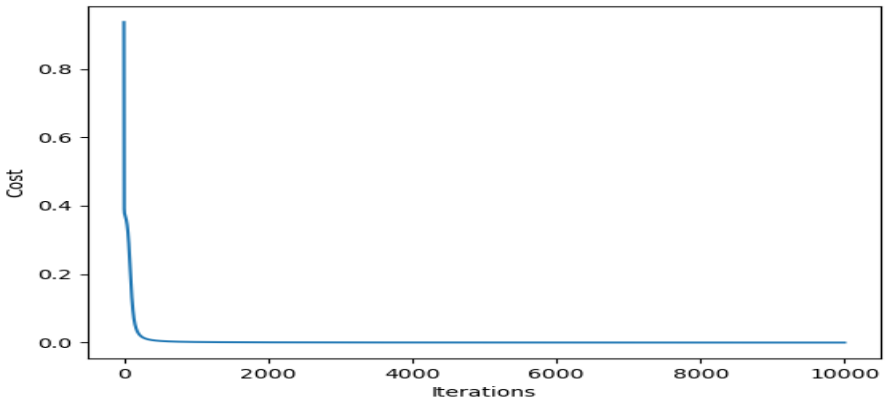


Fig α = 0.9 NOR Logical Function

Alpha	Logical Function	Minimum Cost
0.9	AND	0.00017, on iteration #10000
0.9	NOR	0.00012, on iteration #10000

Task 12

The logistic regression algorithm will firstly assign class 1 as positive class while class 2 and class 3 as negative class. Then algorithm will train and create hypothesis 1 accordingly to specified negative and positive class. Secondly, the algorithm will assign class 1 and class 2 as negative class while class 3 as positive. It will create hypothesis 2 and fit the data accordingly. Finally, it will assign class 1 and class 3 as negative while assigning class 2 as positive. It will then create hypothesis 3 and again fit data accordingly. Now to make predictions amongst our three classifiers the one that predicts the maximim probability the data will be categorized to that positive class.

Whereas, neural networks works extremely well in multi-class classification problems. They can categorize efficiently and effectively with even if there are n classes for classification. Neural networks can have n output neurons representing corresponding classes.

Task 13

Number of Hidden Neurons	Minimum Cost
1	3.01982
2	0.06290
→3	0.03536
5	0.04136
7	0.04028
10	0.03776

We have the minimum cost with hidden neurons = 3. The algorithm has generalized well for all different number of hidden neurons.