Cara Evangeline Chandra Mohan
190539421

MACHINE LEARNING ASSIGNMENT - 1 [Part -2]

1.)Fill out the sigmoid.py function. Now use the plot_sigmoid.py function to plot the sigmoid function. Include in your report the relevant lines of code and the result of the using plot_sigmoid.py.

sigmoid.py

```
from math import e
-
-
output = 1 / (1 + e**(-z))
-
-
```

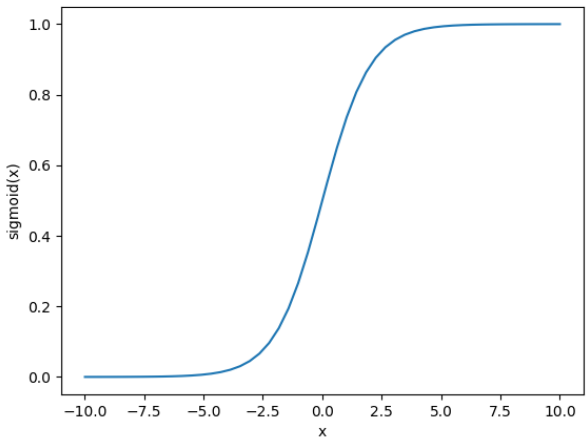


Fig 1: Output of plot_sigmoid.py

The above lines of code were included in sigmoid.py and output of running plot_sigmoid.py is shown in figure 1.

2.)Plot the normalized data to see what it looks like. Plot also the data, without normalization. Enclose the plots in your report.

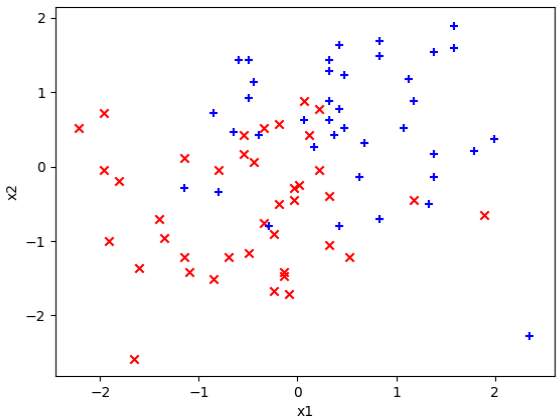


Fig 2:- Normalized Data points

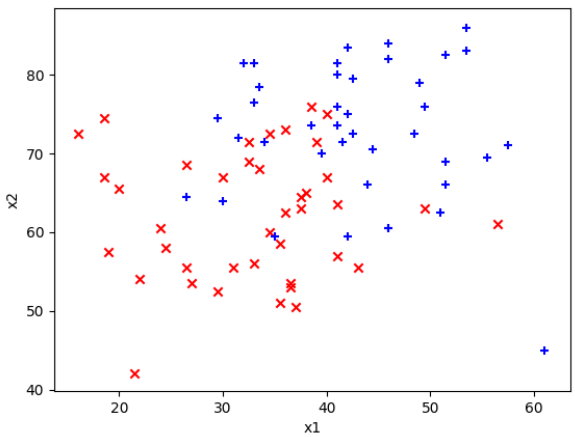


Fig 3:- Without Normalization

When the data is normalized, the x1 and x2 attributes scaling is decreased as shown in figure 2.
When the data is not normalized, x1 and x2 attributes has a wide range of values without proper scaling as shown in figure 3.

Cara Evangeline Chandra Mohan
190539421

3.)Modify the calculate_hypothesis.py function so that for a given dataset, theta and training example it returns the hypothesis. For example, for the dataset X=[[1,10,20],[1,20,30]] and for Theta = [0.5,0.6,0.7], the call to the function calculate_hypothesis(X,theta,0) will return: sigmoid(1 * 0.5 + 10 * 0.6 + 20 * 0.7) The function should be able to handle datasets of any size. Enclose in your report the relevant lines of code.

calculate_hypothesis.py

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

The above code can handle datasets of any size ('i' is the reference to the tuple in a dataset) and can handle any number of variables (ie. attributes) with the help of variable 'j'.

4.)Modify the line "cost = 0.0" in compute_cost.py so that we can use our cost function. To calculate a logarithm you can use np.log(x). Now run the file assgn1_ex1.py. Tune the learning rate, if necessary. What is the final cost found by the gradient descent algorithm? In your report include the modified code and the cost plot.

compute_cost.py

```
cost = (-1 * y[i] * np.log(hypothesis))-((1 - y[i]) * np.log(1-hypothesis))
```

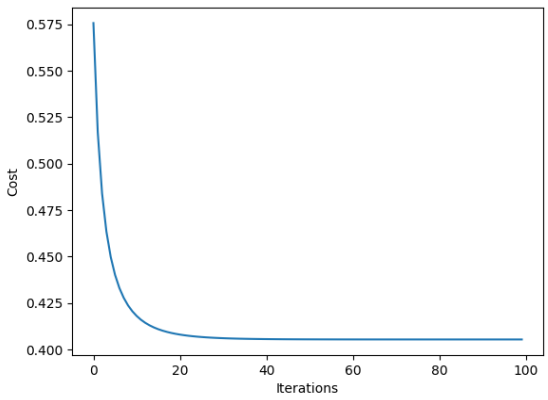


Fig 4:- Cost graph when alpha = 1

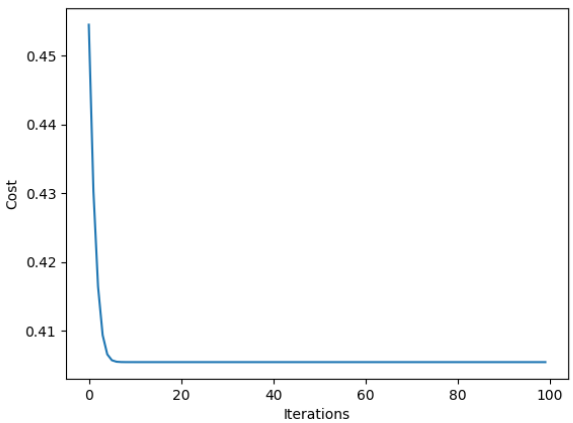


Fig 5:- Cost graph when alpha = 10

For alpha = 1 the final cost found by the gradient descent is 0.40545.
For alpha = 10 the final cost found by the gradient descent is 0.40545.
alpha = 10 is considered better because we get the minimum cost at iteration 37 whereas for alpha = 1 we get the minimum cost at iteration 100.
Figure 4 and 5 shows the output of running assgn1_ex1.py for various values of alpha.

Cara Evangeline Chandra Mohan
190539421

5.)Plot the decision boundary. This corresponds to the line where $\theta^T x = 0$, which is the boundary line's equation. To plot the line of the boundary, you'll need two points of (x_1 ,x_2). Given a known value for x_1 , you can find the value of x_2 . Rearrange the equation in terms of x_2 to do that. Use the minimum and maximum values of x_1 as the known values, so that the boundary line that you'll plot, will span across the whole axis of x_1 . For these values of x_1 , compute the values of x_2 . Use the relevant plot_boundary function in assgn1_ex1.py and include the graph in your report.

plot_boundary.py

```
m = X.shape[0]
for i in range(m):
    if min_x1 > X[i, 1]:
      min_x1 = X[i, 1]
    if max_x1 < X[i, 1]:
      max_x1 = X[i, 1]
x2_on_min_x1 = ((-1 * theta[0]) - (theta[1] * min_x1))/theta[2]
x2_on_max_x1 = ((-1 * theta[0]) - (theta[1] * max_x1))/theta[2]
```

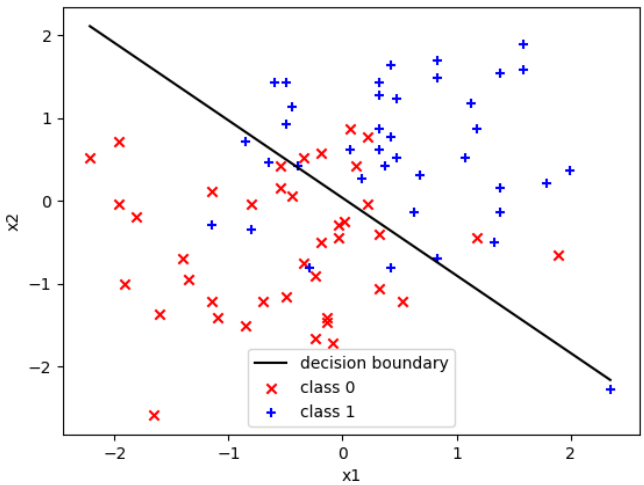


Fig 6:- Graph with decision boundary

Figure 6 shows the output graph when assgn1_ex1.py is run and (min_x1, x2_on_min_x1), (max_x1, x2_on_max_x1) is used to draw the boundary line shown in the graph.

Cara Evangeline Chandra Mohan
190539421

6.)a)Run the code of assgn1_ex2.py several times. In every execution, the data are shuffled randomly, so you'll see different results. Report the costs found over the multiple runs. What is the general difference between the training and test cost? When does the training set generalize well? Demonstrate two splits with good and bad generalisation and put both graphs in your report. b) In assgn1_ex3.py, instead of using just the 2D feature vector, incorporate non-linear features.

a)

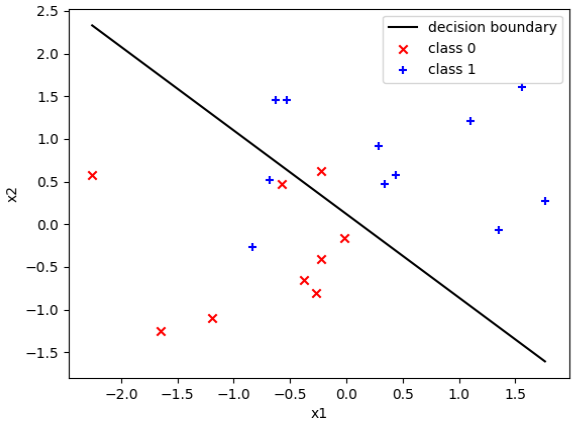
```
jn@jn-Inspiron-5558:~/Downloads/ecs708_assignment1 (1)/ass
Dataset normalization complete.
Dataset normalization complete.
Gradient descent finished.
Final training cost: 0.20424
Minimum training cost: 0.20424, on iteration #100
Final test cost: 0.73984
jn@jn-Inspiron-5558:~/Downloads/ecs708_assignment1 (1)/ass
Dataset normalization complete.
Dataset normalization complete.
Gradient descent finished.
Final training cost: 0.13693
Minimum training cost: 0.13693, on iteration #100
Final test cost: 0.64549
jn@jn-Inspiron-5558:~/Downloads/ecs708_assignment1 (1)/ass
Dataset normalization complete.
Dataset normalization complete.
Gradient descent finished.
Final training cost: 0.51290
Minimum training cost: 0.51290, on iteration #100
Final test cost: 0.43169
```

Fig 7:- Costs for multiple runs

Running assgn1_ex2.py multiple times gives different test and training costs at each run, this is because of the random shuffling of data at every execution.
Gradient descent is obtained using the training data and the weights obtained are applied again on training data, the error that we get now wrt true value is training error and when the weights obtained are applied on test data which gives the test error.
Training set generalizes well when there is a higher degree polynomial order which can fit the training data well.

Good split [train = 60, test = 20]

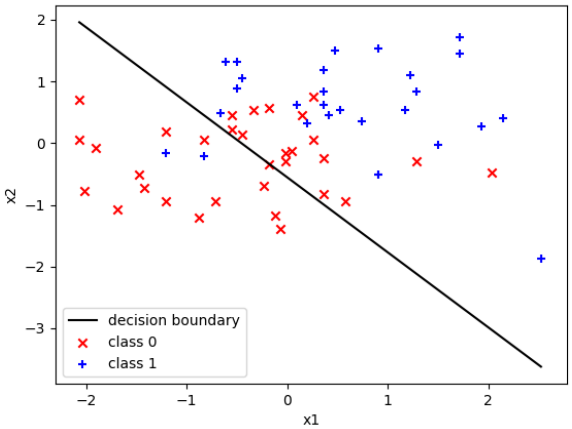
Final training cost: 0.42357
Minimum training cost: 0.42357, on iteration #100
Final test cost: 0.35708



Given graph shows the test dataset classification

Cara Evangeline Chandra Mohan
190539421

Bad split [train = 20, test = 60]



Final training cost: 0.31227
Minimum training cost: 0.31227, on iteration #100
Final test cost: 0.56725

Given graph shows the test dataset classification.

b)

assgn1_ex3.py

```
m = X.shape[0]
a1 = [None] * m
a2 = [None] * m
a3 = [None] * m
for i in range(m):
  a1[i] = X[i, 0] * X[i, 1]
  a2[i] = X[i, 0] * X[i, 0]
  a3[i] = X[i, 1] * X[i, 1]
a1 = np.asarray(a1)
a1 = a1.reshape(X.shape[0],1)
a2 = np.asarray(a2)
a2 = a2.reshape(X.shape[0],1)

a3 = np.asarray(a3)
a3 = a3.reshape(X.shape[0],1)
X = np.append(X, a1, axis=1)
X = np.append(X, a2, axis=1)
X = np.append(X, a3, axis=1)
```

The above lines of code are added to assgn1_ex3.py for including 2D feature vectors to incorporate non-linear features.

a1 adds x1x2 to X
a2 adds x1x1 to X
a3 adds x2x2 to X

Cara Evangeline Chandra Mohan
190539421

7.)Run logistic regression on this dataset. How does the error compare to the one found when using the original features (i.e. the error found in Task 4)? Include in your report the error and an explanation on what happens.

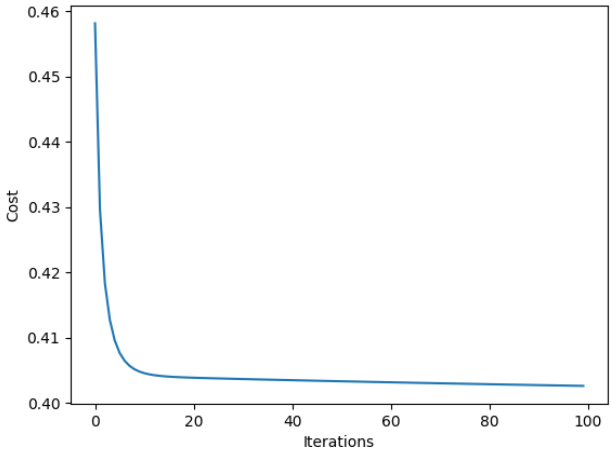


Fig 8: Cost graph for non-linear features with alpha = 1

The minimum cost obtained is 0.40261 for non-linear features for alpha = 1.
The minimum cost obtained for linear features is 0.40545 for alpha = 1.

When more number of features non-linear in nature gets added, the predicted output gets tuned with the true output but since the hypothesis function is sigmoid we don't observe much change in the cost obtained.

8.)a)Experiment with different sizes of training and test set (remember that the total data size is 80) and show the effect of using sets of different sizes by saving the graphs and putting them in your report. b)In the file assgn1_ex5.py, add extra features (e.g. both a second-order and a third-order polynomial) and analyse the effect. What happens when the cost function of the training set goes down but that of the test set goes up?

a)

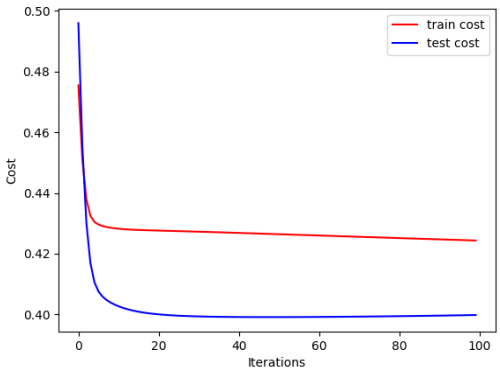


Fig 9: Train = 20 Test = 60

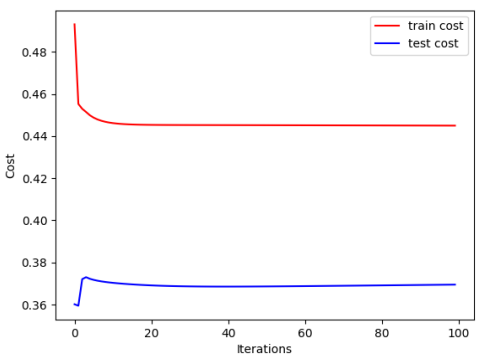


Fig 10: Train = 40 Test = 40

Cara Evangeline Chandra Mohan
190539421

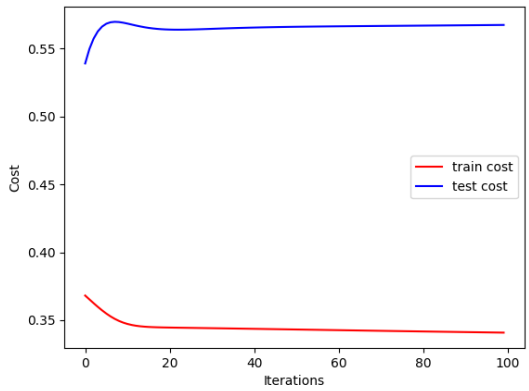


Fig 11: Train = 50 Test = 30

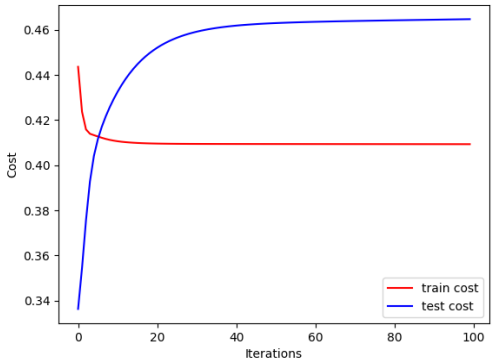


Fig 12: Train = 60 Test = 20

As the training set size is increasing, the test cost is also increasing because of overfitting of data.

b)

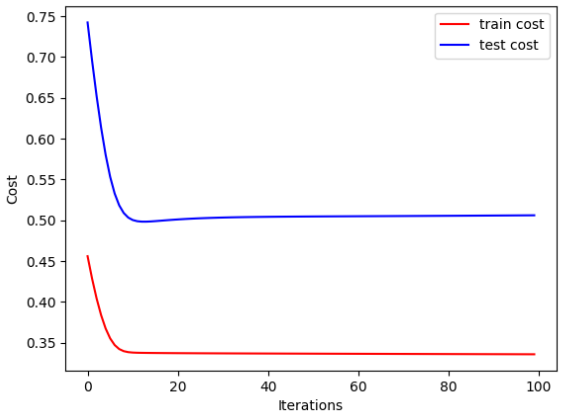


Fig 13: For 2nd and 3rd order polynomial

Figure 13 shows the cost graph for training set of size 40 and test set of size 40 for a polynomial of order 2 and 3. When the polynomial order increases, the training dataset is memrized and trained very well which leads to increase in test dataset.

9)With the aid of a diagram of the decision space, explain why a logistic regression unit cannot solve the XOR classification problem.

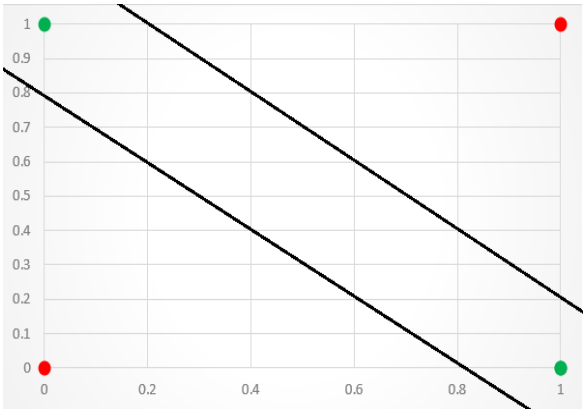
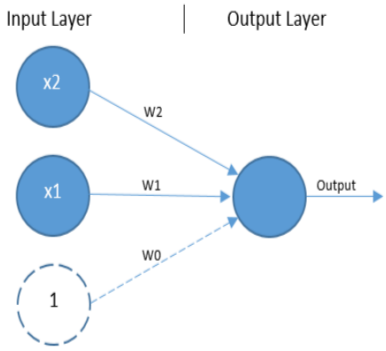


Fig 14:- Decision space

Cara Evangeline Chandra Mohan
190539421



XOR Classification problem is not linearly separable. When x1 and x2 is given as input to the perceptron, the output will be sigmoid(w0+w1x1+w2x2) and the problem given must be linearly separable to get the correct output and hence a logistic regression unit cannot solve XOR Problem as figure 14 indicates that XOR is not linearly separable.

10)Implement backpropagation's code, by filling the backward_pass() function, found in NeuralNetwork.py. Although XOR has only one output, your implementation should support outputs of any size.

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):
    output_deltas[i] = (outputs[i] - targets) * sigmoid_derivative(outputs[i])
# Step 2. Hidden deltas are used to update the weights of the hidden layer
hidden_deltas = np.zeros((len(self.y_hidden)))
for i in range(len(hidden_deltas)):
    for j in range(len(output_deltas)):
        hidden_deltas[i] = hidden_deltas[i] + output_deltas[j] * self.w_out[i, j]
    hidden_deltas[i] = hidden_deltas[i] * sigmoid_derivative(self.y_hidden[i])
# Step 3. update the weights of the output layer
for i in range(len(self.y_hidden)):
    for j in range(len(output_deltas)):
        self.w_out[i, j] = self.w_out[i, j] - (n * output_deltas[j] * self.y_hidden[i])
hidden_deltas = hidden_deltas[1:]
# Step 4. update the weights of the hidden layer
for i in range(len(inputs)):
    for j in range(len(hidden_deltas)):
        self.w_hidden[i, j] = self.w_hidden[i, j] - (n * hidden_deltas[j] * inputs[i])
```

Cara Evangeline Chandra Mohan
190539421

11)Change the training data in xor.m to implement a different logical function, such as NOR or AND. Plot the error function of a successful trial.

AND Gate:

Sample #01 | Target value: 0.00 | Predicted value: 0.00122
Sample #02 | Target value: 0.00 | Predicted value: 0.00820
Sample #03 | Target value: 0.00 | Predicted value: 0.00821
Sample #04 | Target value: 1.00 | Predicted value: 0.99069
Minimum cost: 0.00011, on iteration #10000

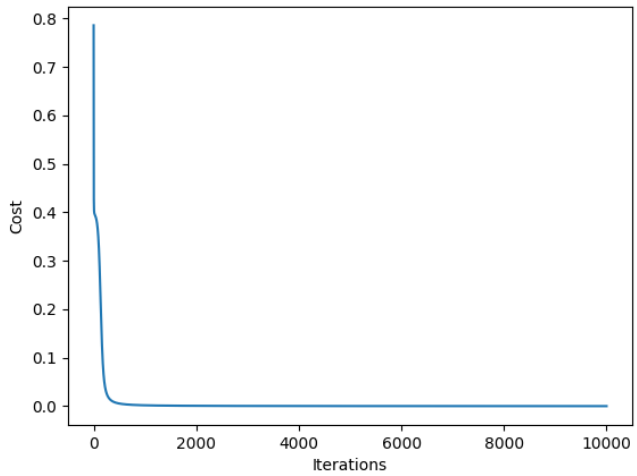


Fig 15 :- Cost graph of AND Gate

NOR Gate:

Sample #01 | Target value: 1.00 | Predicted value: 0.99003
Sample #02 | Target value: 0.00 | Predicted value: 0.00603
Sample #03 | Target value: 0.00 | Predicted value: 0.00653
Sample #04 | Target value: 0.00 | Predicted value: 0.00144
Minimum cost: 0.00009, on iteration #10000

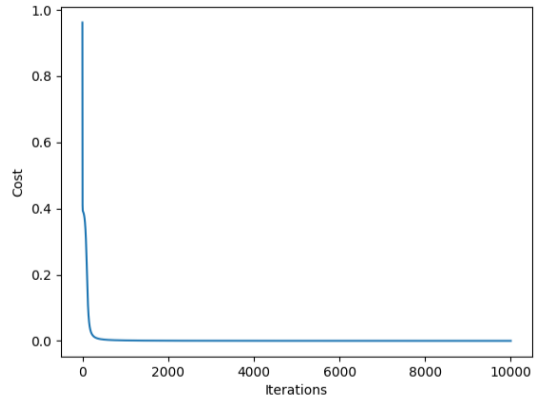


Fig 16:- Cost graph of NOR Gate

Cara Evangeline Chandra Mohan
190539421

12)The Iris data set contains three different classes of data that we need to discriminate between. How would you accomplish this if we used a logistic regression unit? How is this scenario different, compared to the scenario of using a neural network?

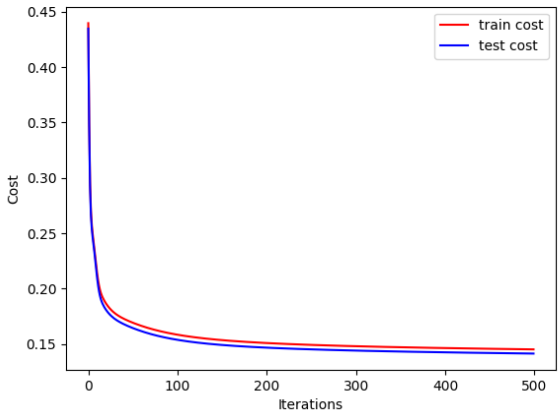
A logistic regression classifier has to be trained for each class i to predict the probability that y = i. On a new input x to make a prediction, pick the class i that maximizes the logistic regression classifier.

In neural network we will have 3 output layers out of which one output layer will have it's output as 1 and other two output layers will have it's output as 0 which indicates that a new tuple feeded to a neural network belongs to the class which has ouput 1.

13)Run irisExample.py using the following number of hidden neurons: 1, 2, 3, 5, 7, 10. The program will plot the costs of the training set (red) and test set (blue) at each iteration. What are the differences for each number of hidden neurons? Which number do you think is the best to use? How well do you think that we have generalized?

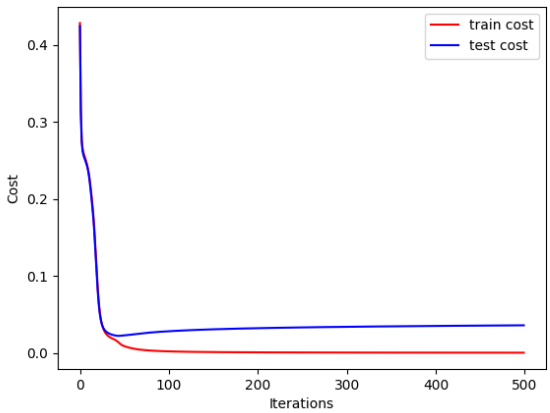
Hidden Neurons = 1

Minimum cost: 2.76482, on iteration #500



Hiddens Neurons = 2

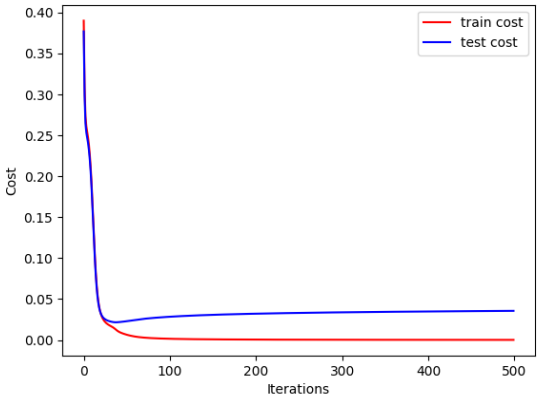
Minimum cost: 0.00556, on iteration #500



Cara Evangeline Chandra Mohan
190539421

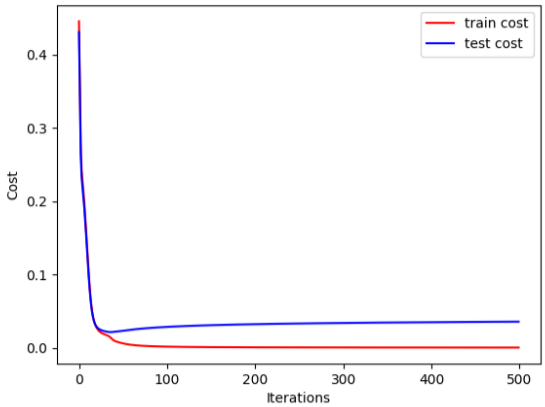
Hidden Neurons = 3

Minimum cost: 0.00460, on iteration #500



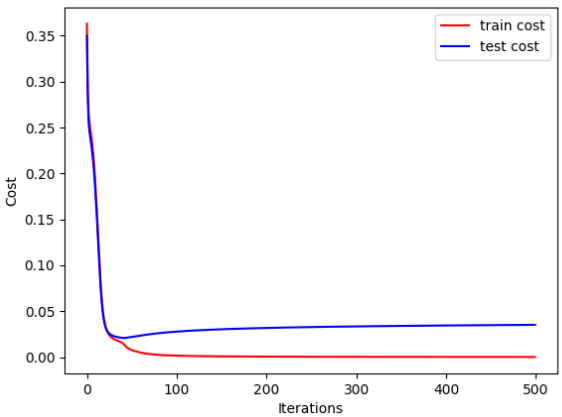
Hidden Neurons = 5

Minimum cost: 0.00432, on iteration #500



Hidden Neurons = 7

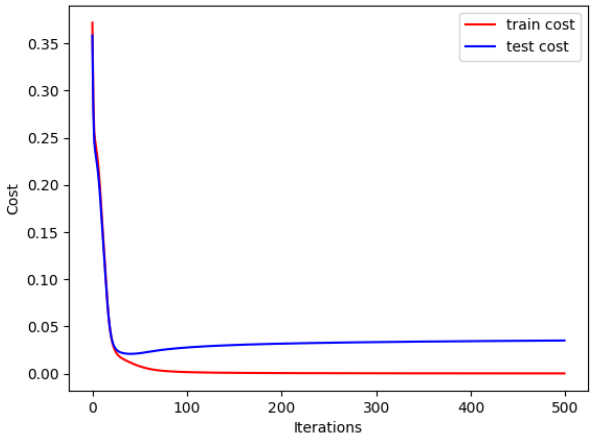
Minimum cost: 0.00426, on iteration #500



Cara Evangeline Chandra Mohan
190539421

Hidden Neurons = 10

Minimum cost: 0.00350, on iteration #500



When number of hidden neurons is 10, the program gives the least test error and hence 10 hidden neurons is considered to be the best.