ECS708P - MACHINE LEARNING - 2017/18

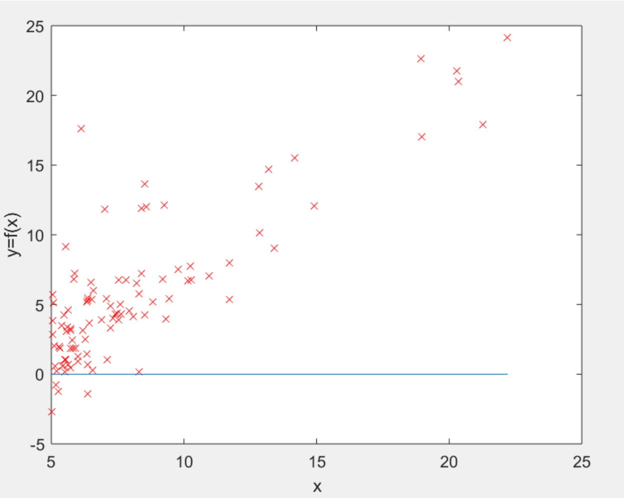
ASSIGNMENT 1 (LINEAR REGRESSION + LOGISTIC REGRESSION)

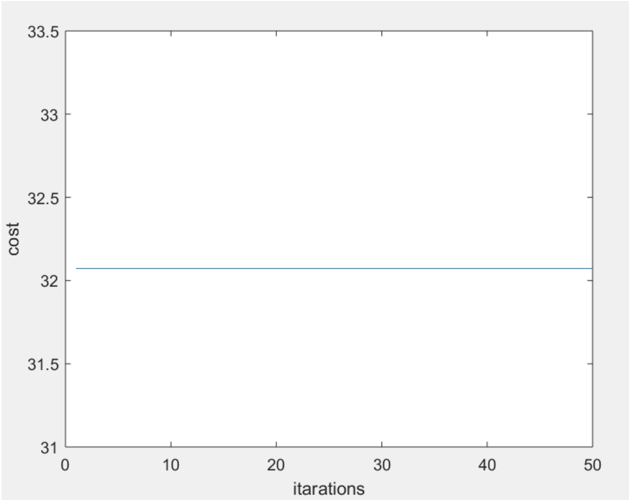
Sarang Kharche - 150740395 11/14/17 Machine Learning

Table of Contents

In this exercise, we have one variable, X, and we building a model to predict one

Hypothesis and Cost Graph are flat as the hypothesis is not calculated yet.





Assignment 1: Part 1 - Linear Regression

1- Linear Regression with One Variable

Task 1:

Modify the function *calculate_hypothesis.*m to return the predicted value for a single specified training example. Include in the report the corresponding lines from your code.

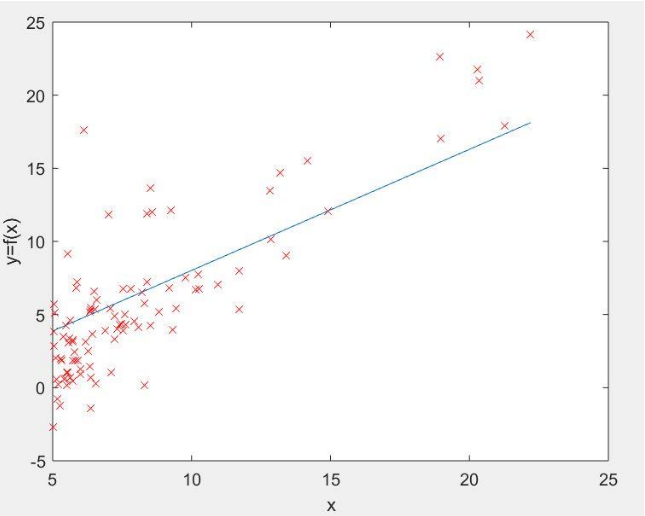
Calculate Hypothesis:

```
function hypothesis = calculate_hypothesis (X, theta,
training_example)

%CALCULATE_HYPOTHESIS This calculates the hypothesis for a given X,
%theta and specified training example

hypothesis = X(training_example,1) *theta (1) +
X(training_example,2) *theta (2);

end
```

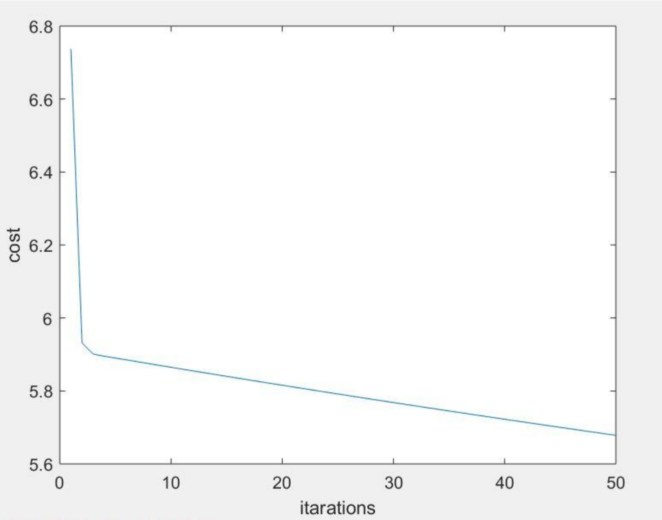
After making changes to the hypothesis and adding this line of code

```
hypothesis = X(training_example,1) *theta (1) +
X(training_example,2) *theta (2);
```

The hypothesis graph below shows the changes and not the flat line anymore.



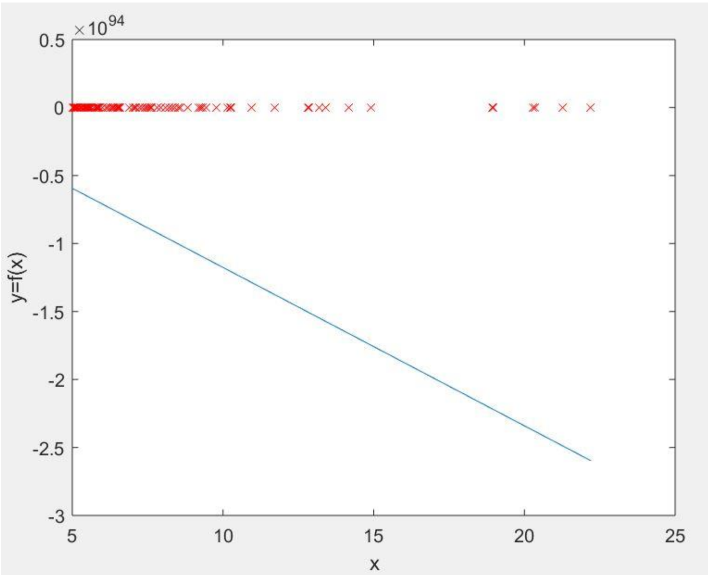
When now we run mllab1.m we can see the cost going down overtime.

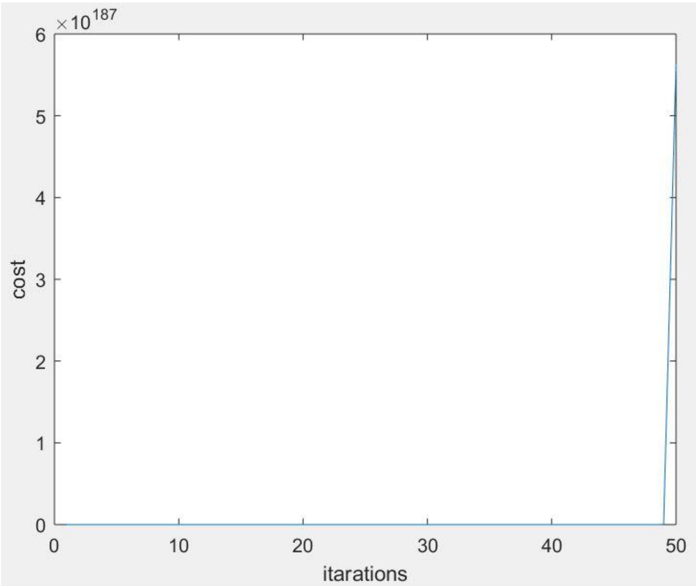


Now we modified the values for the learning rate, alpha in mllab1.m and the results are posted below.

First, we will increase the value of alpha to 0.90.

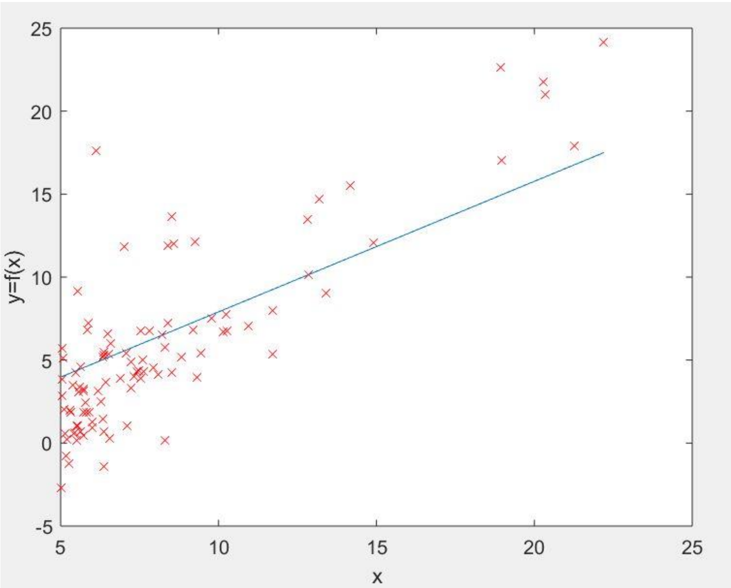
```
alpha = 0.90;
```

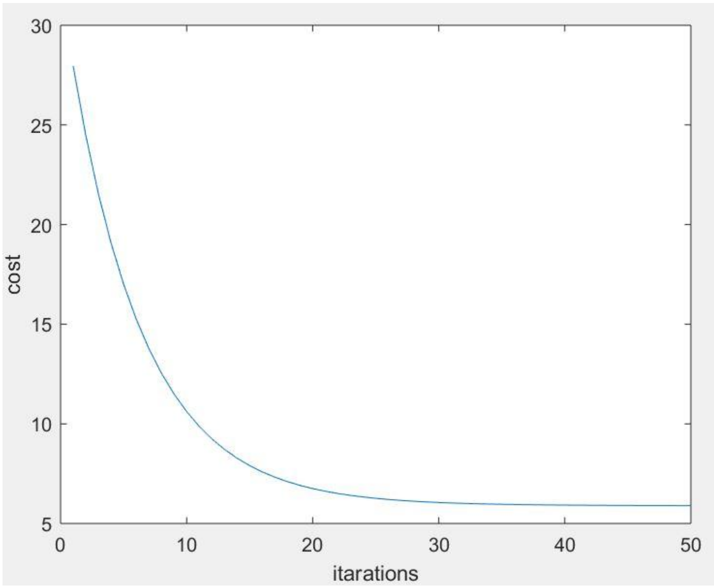




Secondly, we will decrease the value to 0.001.

```
alpha = 0.001;
```





2. Linear Regression with Multiple Variables

Task 2:

Modify the functions *calculate_hypothesis* and *gradient_descent* to support the new hypothesis function. This should be sufficiently general so that we can have any number of extra variables. Include the relevant lines of the code in your report.

Following is the *calculate_hypothesis,* a few lines of code were added as shown below.

```
function hypothesis = calculate_hypothesis (X, theta,
training_example)

%CALCULATE_HYPOTHESIS This calculates the hypothesis for a given X,

%theta and specified training example


hypothesis = sum (X (training_example, :) * theta');


end
```

After making changes to the hypothesis and adding this line of code
`hypothesis = sum (X (training_example, :) * theta');`

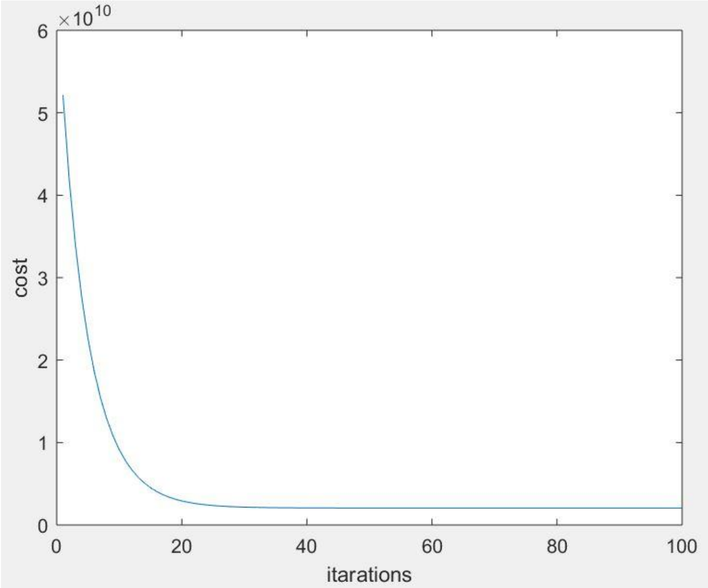
Following is the code that was added to *gradient_descent* to support the new hypothesis function, this was added to get the output for theta_0 = theta(1); theta_1 = theta(2); theta_2 = theta(3);

```
%update theta(1) and store in temporary variable theta_0

sigma = 0.0;
for i = 1:m
hypothesis = X(i, 1) * theta(1) + X(i, 2) * theta(2);
output = y(i);
sigma = sigma + (hypothesis - output);
end

theta_0 = theta_0 - ((alpha * 1.0) / m) * sigma;
```

The theta was then updates as below:

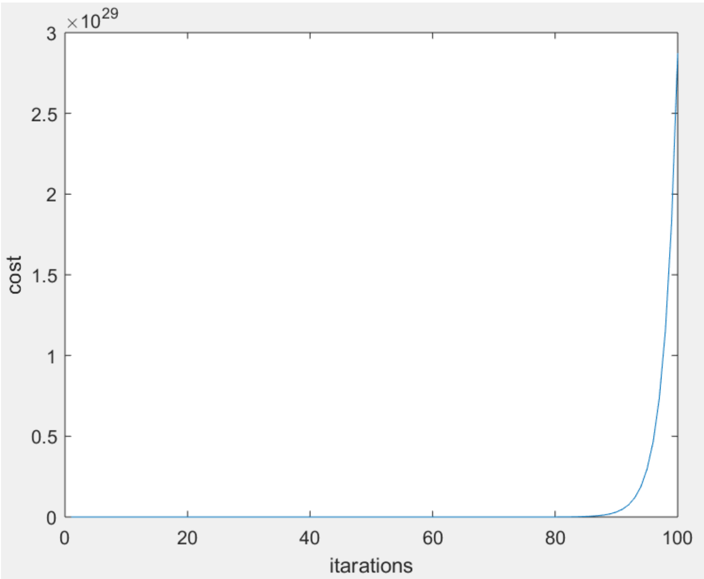
```
theta = [theta_0, theta_1, theta_2];
```

Following is the graph result for the hypothesis (The price has fallen over time)

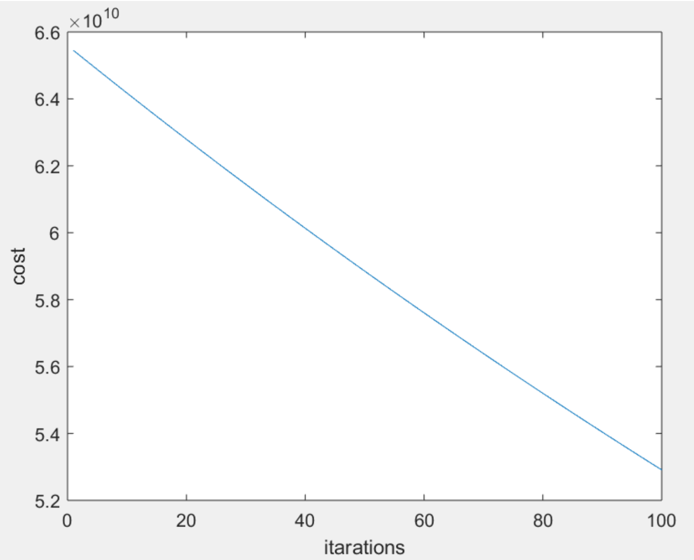


Now we will see how different values of alpha affect the convergence of the algorithm. The results are as posted below.

```
alpha = 0.90;
```



```
alpha = 0.001;
```



How much does your algorithm predicts that a house with 1650 sq. ft. and 3 bedrooms cost?

We use the formula for normalization and came up with the following line of code.

```
p_1 = (1650 - mean_vec(1))/std_vec(1);
r_1 = (3 - mean_vec(1))/std_vec(1);
```

To predict the value, we are using the following formula which will generate the results.

```
q_1 = t(1)+p_1*t(2)+r_1*t(2);
```

The results are as follows:

2.9758e+04

How about 3000 sq. ft. and 4 bedrooms?

We use the formula for normalization and came up with the following line of code.

```
a_1 =  (3000 - mean_vec(1))/std_vec(1);
b_1 = (4 - mean_vec(1))/std_vec(1);
```

To predict the value, we are using the following formula which will generate the results.

```
z_1 = t(1)+a_1*t(2)+b_1*t(2);
```

The results are as follows:

2.0847e+05

3. Regularized Linear Regression

Task 3:

Note that the punishment for having more terms is not applied to the bias. This cost function has been implemented already in the function compute_cost_regularised. Modify gradient_descent to use the compute_cost_regularised method instead of compute_cost. Include the relevant lines of the code in your report and a brief explanation.

Modifying *gradient_descent* to create:

```
theta_0 = theta(1);
theta_1 = theta(2);
theta_2 = theta(3);
theta_3 = theta(4);
theta_4 = theta(5);
theta_5 = theta(6);
```

Update the theta for the new theta values.

```
%update theta
theta = [theta_0, theta_1, theta_2, theta_3, theta_4, theta_5];
```

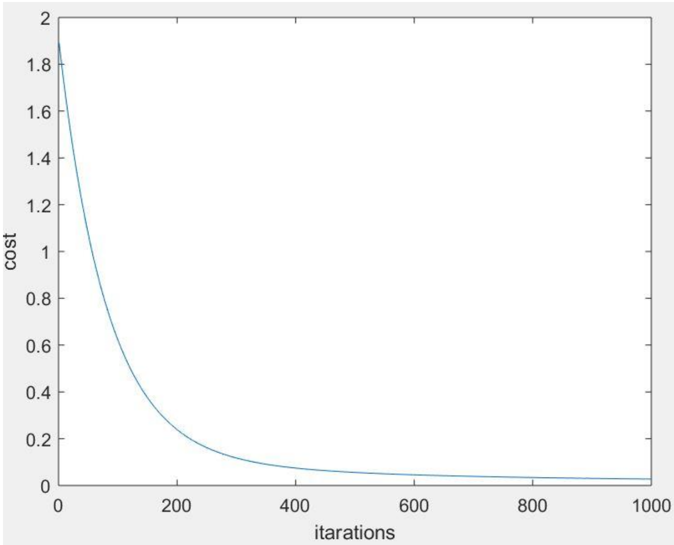
Update the cost_vector to use compute_cost_regularised instead of compute_cost

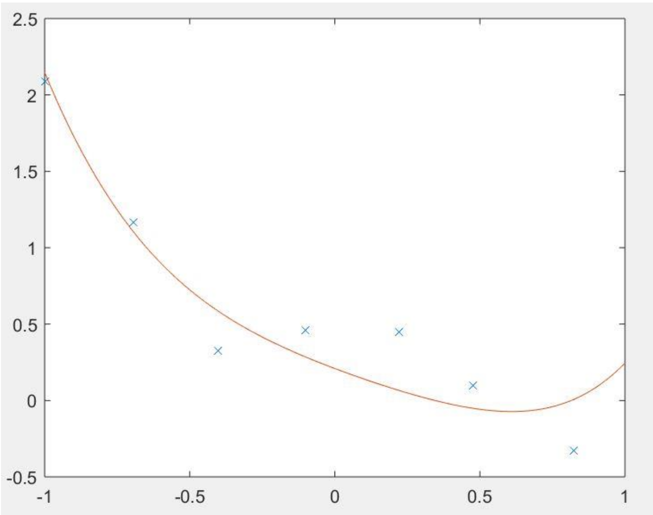
```
%update cost_vector
cost_vector =[cost_vector;compute_cost_regularised(X, y, theta, l)];
```

Calculate_hypothesis is modified to run mllab3.m

```
function hypothesis = calculate_hypothesis(X, theta, training_example)
   %CALCULATE_HYPOTHESIS This calculates the hypothesis for a given X,
   %theta and specified training example
   hypothesis = sum(X(training_example,:)* theta');
end
```

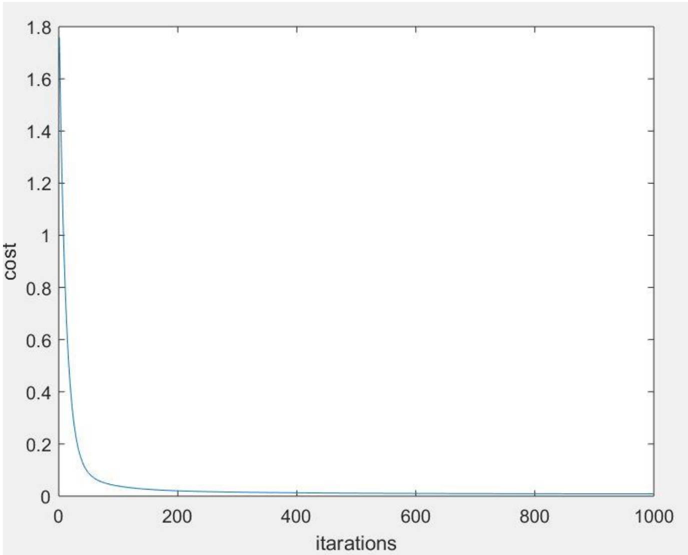
Plot the hypothesis function found at the end of the optimization.

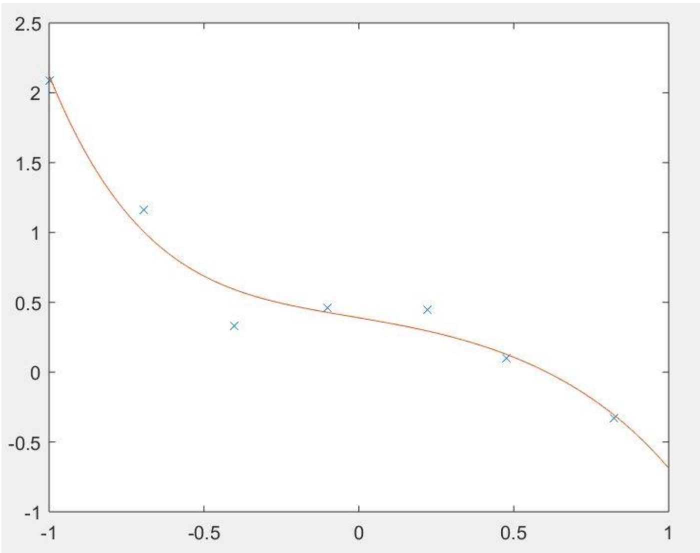




After a few attempts to find the best value for alpha, I am came up with 0.007

The results are as posted below.





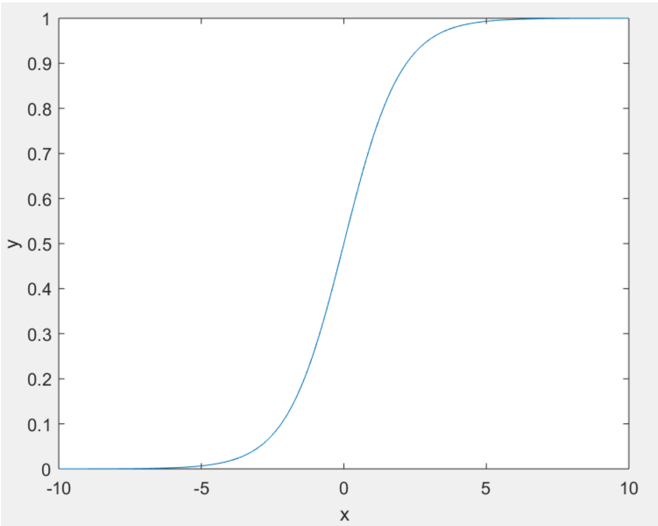
Assignment 1 – Part 2: Logistic Regression and Neural Networks

Task 1:

Include in your report the relevant lines of code and the result of the running the plot_sigmoid_function.m.

plot_sigmoid_function.m

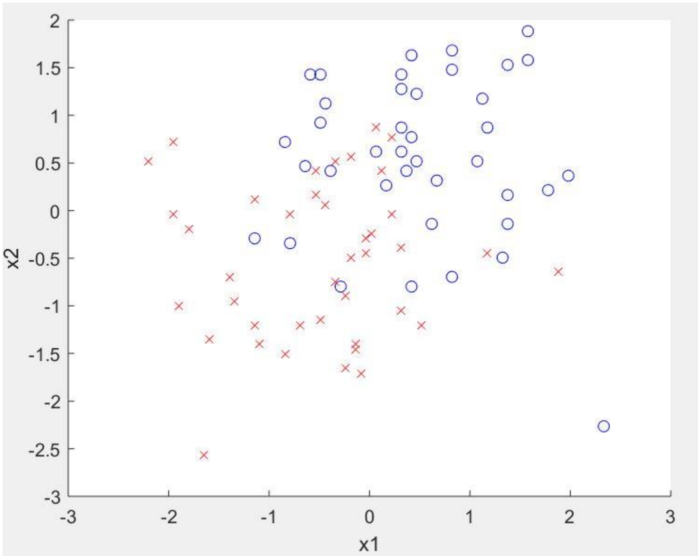
```
h=plot_sigmoid;
```

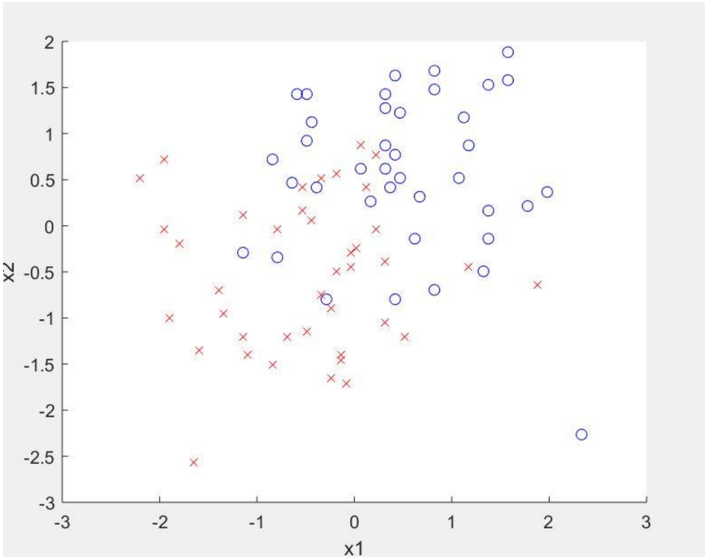


Task 2:

Plot the data again to see what it looks like in this new format. Enclose this in your report.

```
% this loads our data
[X,y] = load_data_ex1();
% now we want to normalise our data
[X,mean,std] = normalise_features(X);
% after normalising we add the bias
X=[ones(size(X,1),1) X];
h=plot_data_function(X,y);
```





1.1. Cost function and gradient for logistic regression

Task 3:

Modify the calculate_hypothesis so that for a given dataset, theta and training example it returns the hypothesis.

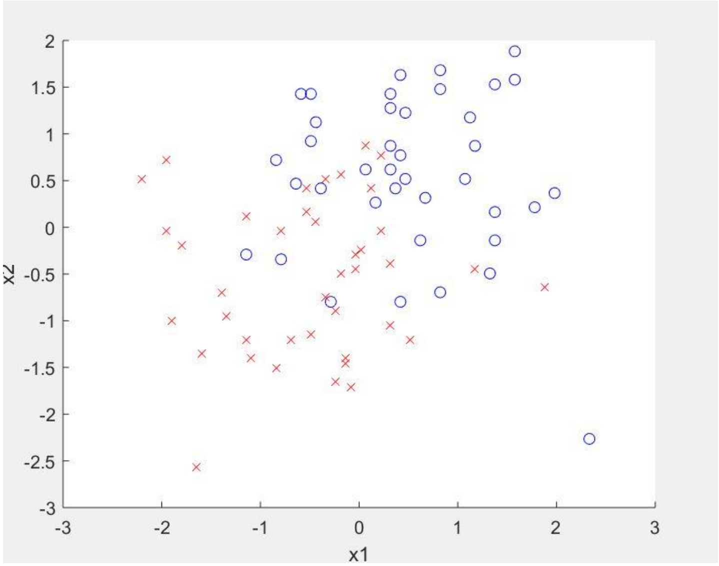
Modifying the calculate_hypothesis: to:

```
function result=calculate_hypothesis(X,theta,training_example)

  hypothesis = 0.0;

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %Calculate the hypothesis for the i-th training example in X.

  hypothesis = X(training_example,1) *theta(1) + X(training_example,2)*theta(2);
  %hypothesis

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  result=sigmoid(hypothesis);
```

```
%END OF FUNCTION
```

Modifying the sigmoid.m

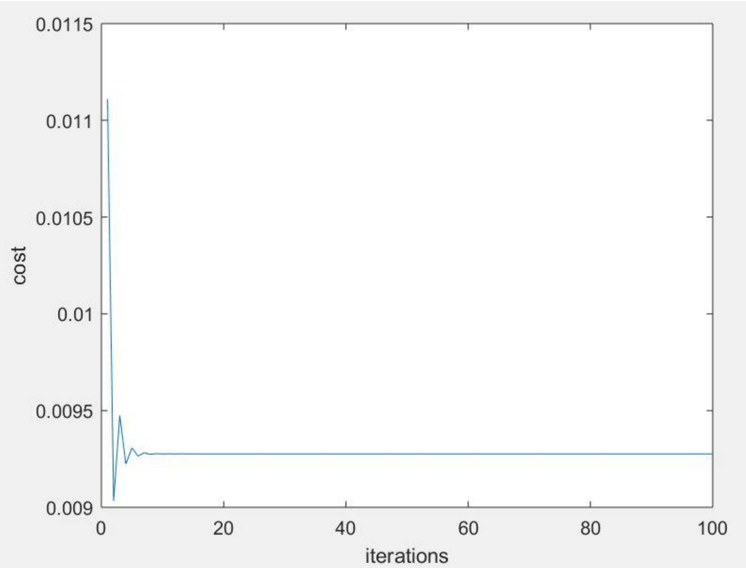
```
function output=sigmoid(z)

   % modify this to return z passed through the sigmoid function

   %%%%%%%%%%%%%%%%%%%%%%%%%

   output = zeros(size(z));
  output = 1 ./ (1 + exp(-z));
   %%%%%%%%%%%%%%%%%%%%%%%%%

%END OF FUNCTION
```



Task 4:

Modify the line: cost = 0.0 in compute_cost(X,y,theta so that it uses the cost function:

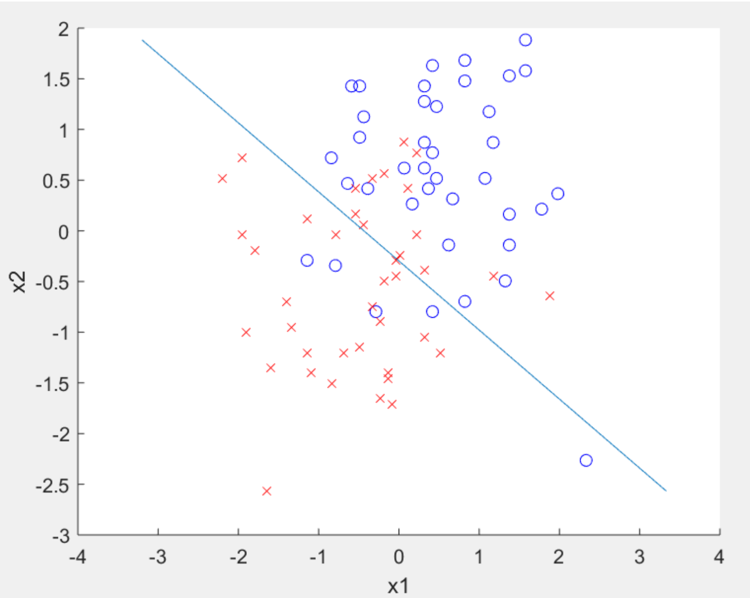
```
for i=1:m
hypothesis = calculate_hypothesis(X,theta,i);
output = y(i);

cost = -output*log(hypothesis)-(1-output)*log(1-hypothesis);

J = 1 / m * sum(-y .* log(hypothesis) - (1 - y) .* log(1 - hypothesis));

% modify this to calculate the cost function, using hypothesis and output
cost = 0.0;
J =J+cost;

end
```

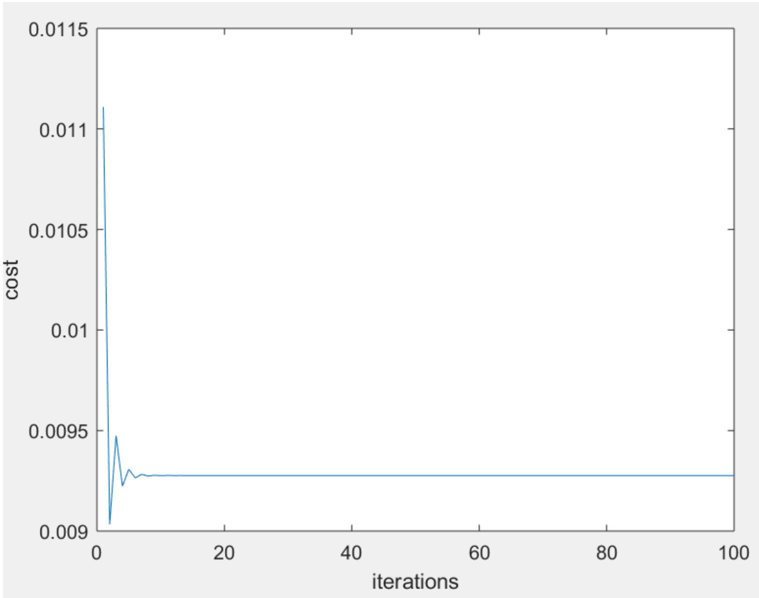


Task 5:

Plot the decision boundary. This corresponds to the line where $\theta T\mathbf{x} = 0$ For example, if $h\theta = \theta 1x1 + \theta 2x2$ the boundary is where $\theta 1x1 + \theta 2x2 =$

```
% plot our data and decision boundary
  plot_data_function(X,y)
  plot_boundary(X,t)
```

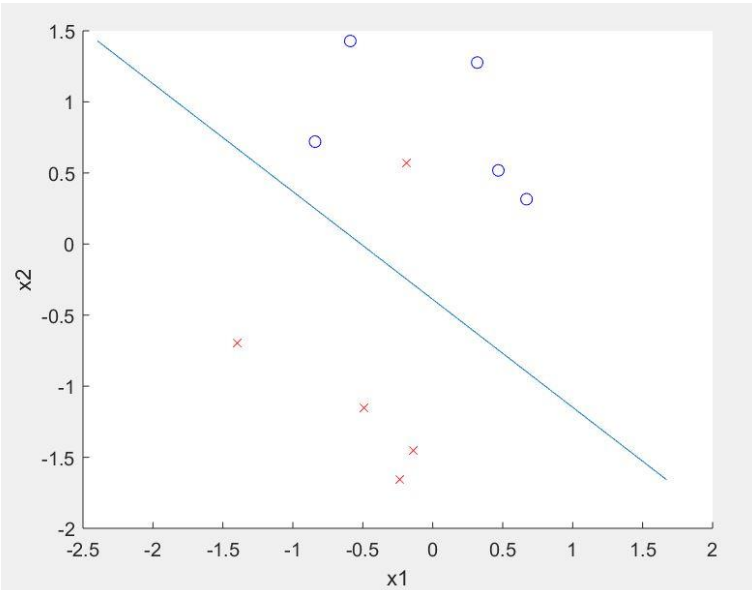


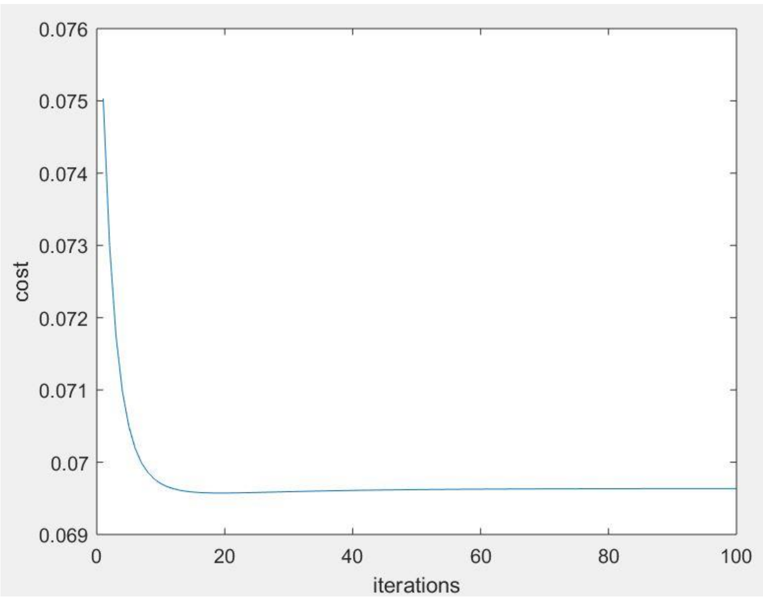


Task 6:

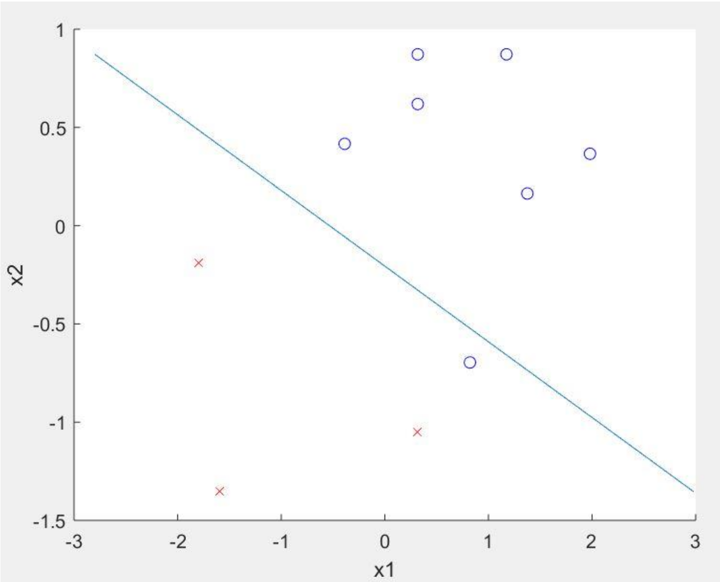
Run the code in lab2_lr_ex2.m several times. What is the general difference between the training and test error? When does the training set generalize well? Demonstrate two splits with good and bad generalisation and put both graphs in your report.

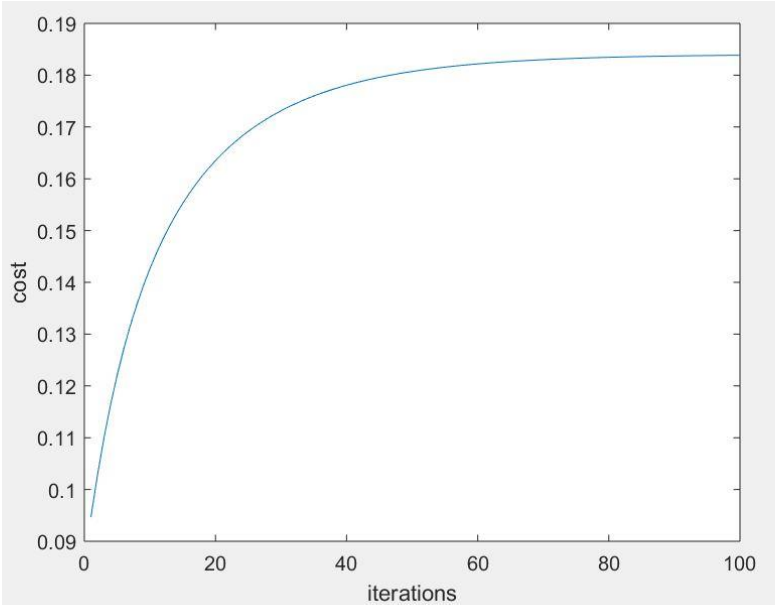
Training error:0.069635



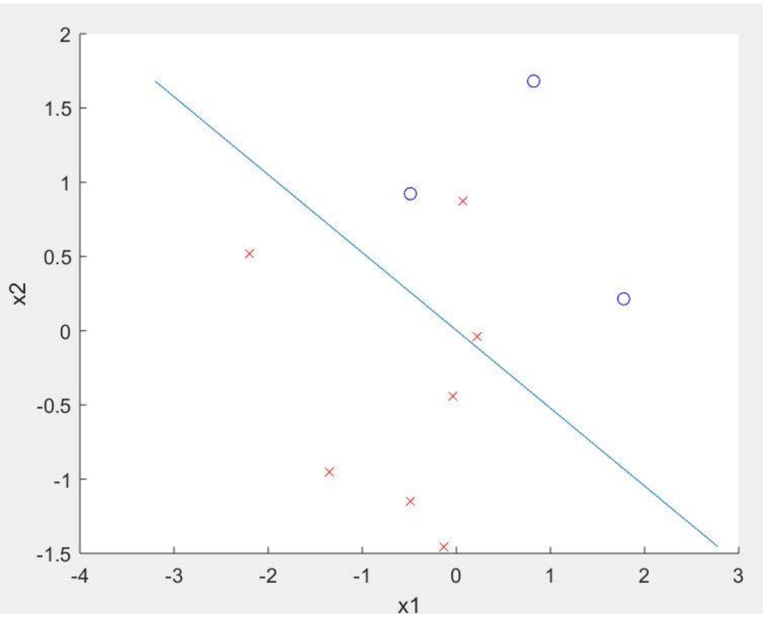


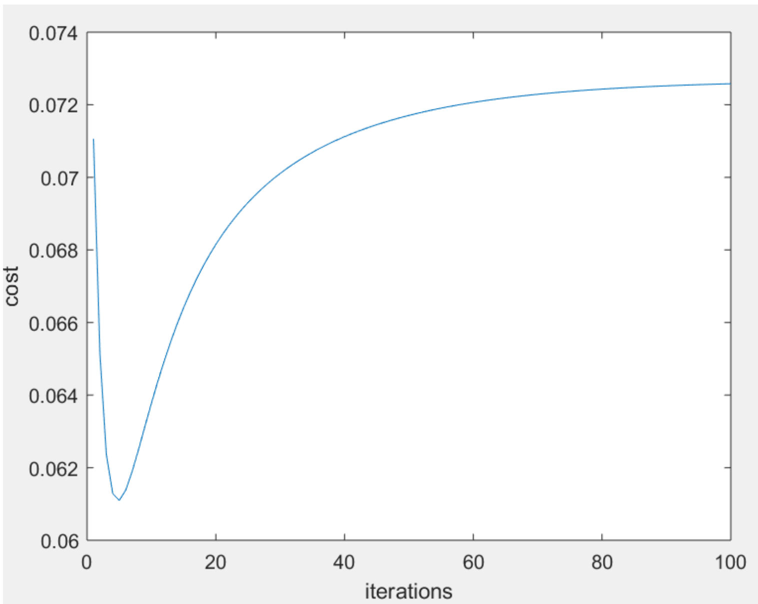
Training error:0.18386





Training error:0.072579

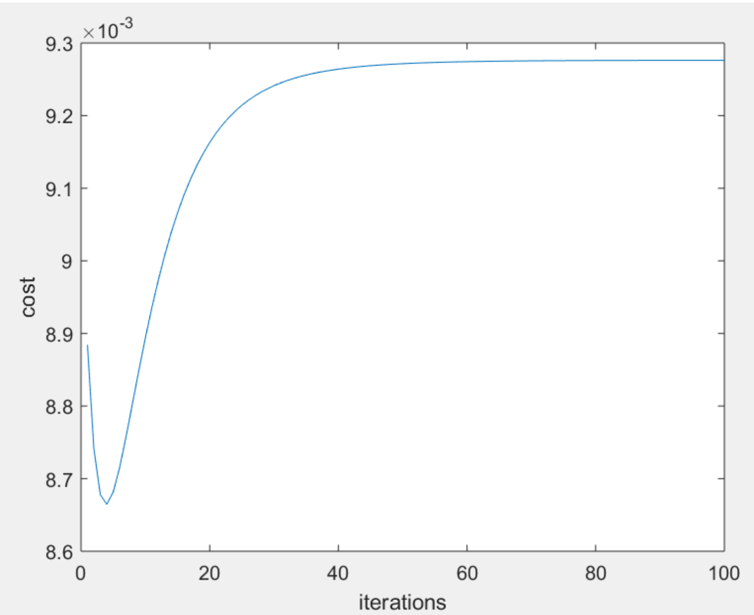




Task 7:

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

```
% after normalising we add the bias
X=[ones(size(X,1),1),X];
theta=ones(1,6);

% for question 7, modify the dataset X to have more features (in each row)

% append to X(i),the following features:

% here append x_2 * x_3 (remember that x_1 is the bias)

X(:,4) = X(:,2).*X(:,3);


% here append x_2 * x_2 (remember that x_1 is the bias)

X(:,5) = X(:,2).^2;

%here append x_3 * x_3 (remember that x_1 is the bias)
%X(:,6) = X(:,3).*X(:,3);
X(:,6) = X(:,3).^3;
```



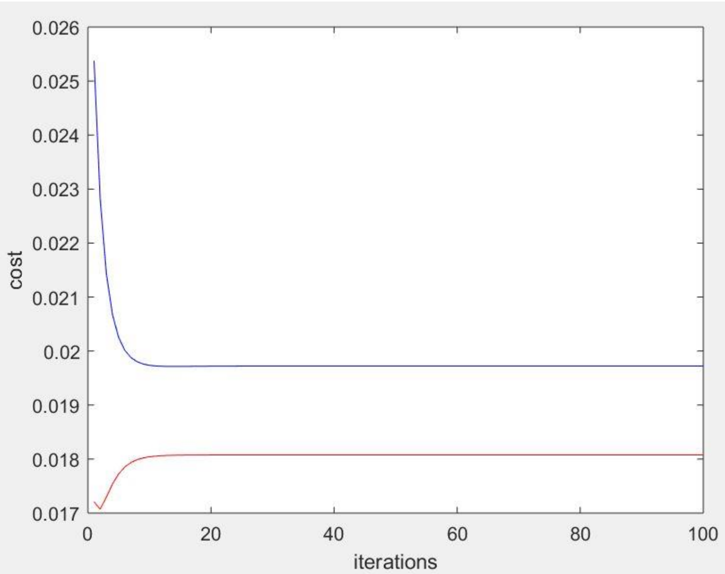
Task 8:

In lab2_lr_ex3b.m the data is split into a test set and a training set. Add your new features from the question above. Modify the function gradient_descent_training () to store the current cost for the training set and testing set. Store the cost of the training set to cost_array_training and for the test set to cost_array_test

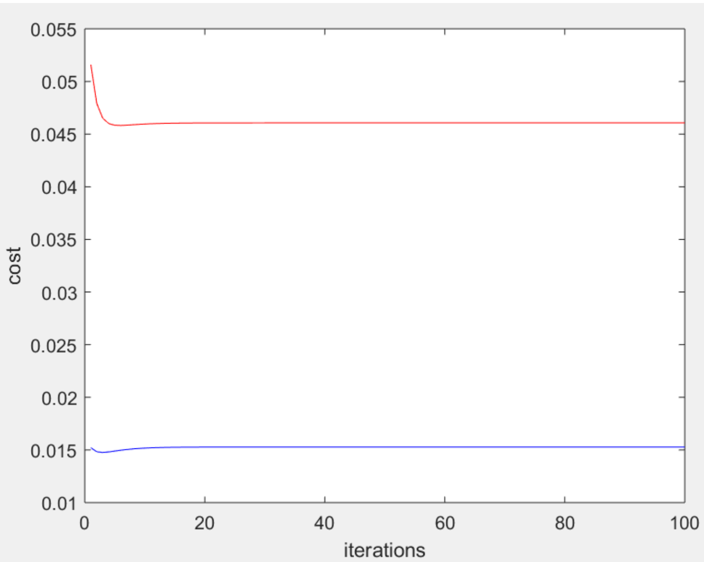
Gradient_descent_training

```
% update cost_array
% add code here: to update cost_array_training and cost_array_test
  cost_array_training(it)=compute_cost(X, y, theta);
  cost_array_test(it)= compute_cost(test_X,test_y,theta);
```

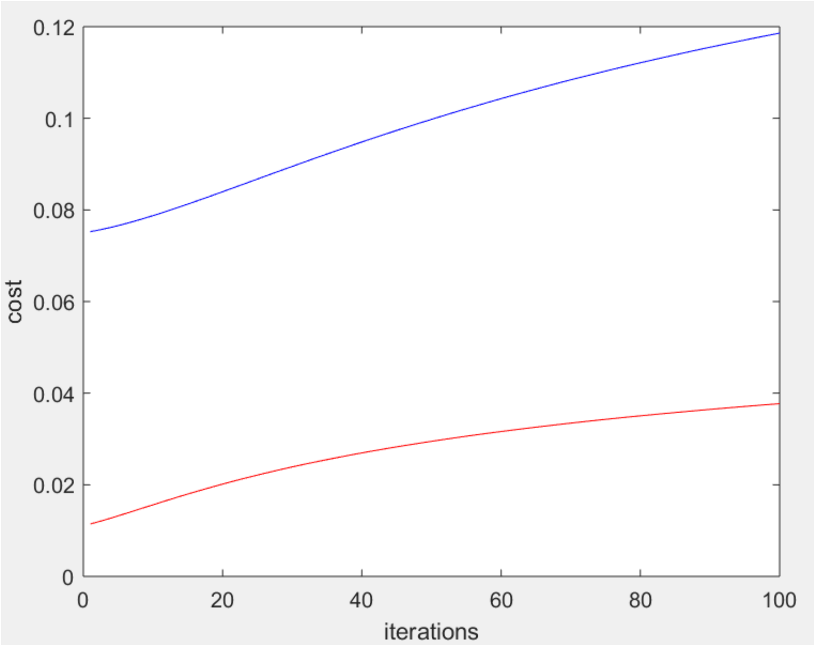
Data Size: 40, Training:0.019723, Test:0.018076



Data Size: 60, Training: 0.015277, Test: 0.046065



`Data Size: 10, Training: 0.11859, Test: 0.037704`



Task 9:

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

To solve the problem, we need to introduce a new layer into our neural networks. This layer, often called the 'hidden layer', allows the network to create and maintain internal representations of the input. Here's is a network with a hidden layer that will produce the XOR truth table above: XOR Network.[i]

2. Neural Network

Task 10.

Implement backpropagation. Although XOR only has one output, this should support outputs of any size. Do this following the steps below.

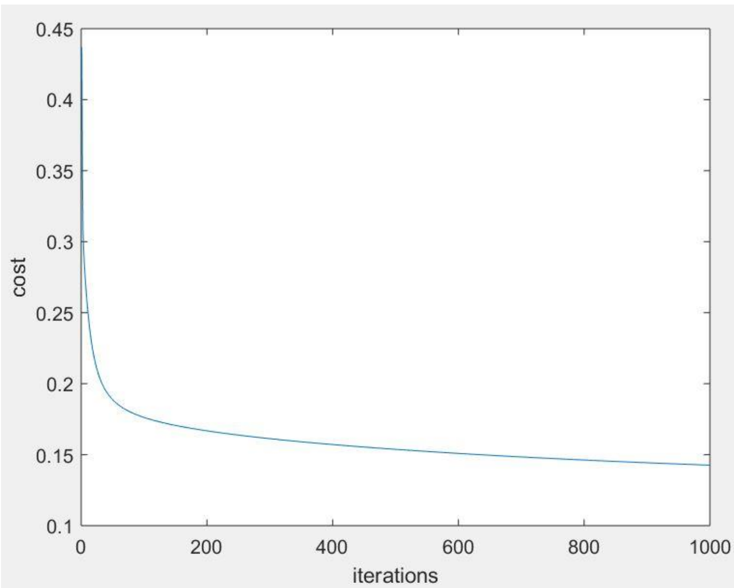
Modified Sigmoid.m

```
function  sigmoid_output=sigmoid(z)
% change this to apply the sigmoid to the data below:

sigmoid_output  = 1.0 ./ (1.0 + exp(-z));

%sigmoid_output = 0.0;
end
```

Modified NeuralNetwork.m

```
% Step 1. Output deltas (used to change weights from hidden --> output)
            output_deltas = zeros(1,length(nn.output_neurons));
            outputs=nn.output_neurons;
            for i=1:length(outputs)
                output_deltas(i) = (outputs(i)-
targets(i))*sigmoid_derivative(outputs(i));
            end
            % Step 2. Hidden deltas (used to change weights from input -->
output).
            hidden_deltas = zeros(1,length(nn.hidden_neurons));
            % hint... create a for loop here to iterate over the hidden
neurons and for each

            %###########################   my coe is in between
##########################################
            % hidden neuron create another for loop to iterate over the
ouput neurons

            %for i=1:length(nn.hidden_neurons)
            %hiddens = nn.hidden_neurons;
              %for j=1:length(hiddens)
              %hidden_deltas =
sigmoid_derivative*(inputs(j)+(inputs(j))*(outputs));
              %end
            %end
            %###########################   my coe is in between
##########################################


            % Step 3. update weights output --> hidden
            for i=1:length(nn.hidden_neurons)
                for j=1:length(output_deltas)
                    nn.output_weights(i,j) =nn.output_weights(i,j) -
(output_deltas(j) * nn.hidden_neurons(i) * learning_rate);
                end
            end
```

```
            % here we are removing the bias from the hidden neurons as
there is no
            % connection to it from the layer below
            hidden_deltas = hidden_deltas(2:end);

            % Step 4. update weights input --> hidden.
            % hint, use a similar process to step 3, except iterate over
the input neurons and hidden deltas

            %for i=1:length(nn.hidden_neurons)
                for i=1:length(output_deltas)
                    nn.output_weights(i) =nn.output_weights(i) -
(output_deltas(i) * nn.hidden_neurons(i) * learning_rate);
                end
            %end


            % this is our cost function
            J = 0.0;
            for t =1:length(targets)
                J = J + 0.5*(nn.output_neurons(t)-targets(t))^2;
            end
        end

        function propagation=forward_propagate(inputs,nn)
            %
            %Calculates the output by feeding the inputs forward through
the network
            %
            nn.reset_activations()
            % activate hiddens_neurons
            for i=1:length(nn.hidden_neurons)
                hidden_neuron = 0.0;
                for j=1:length(inputs)
                    hidden_neuron =hidden_neuron + inputs(j) *
nn.hidden_weights(j,i);
                end
```

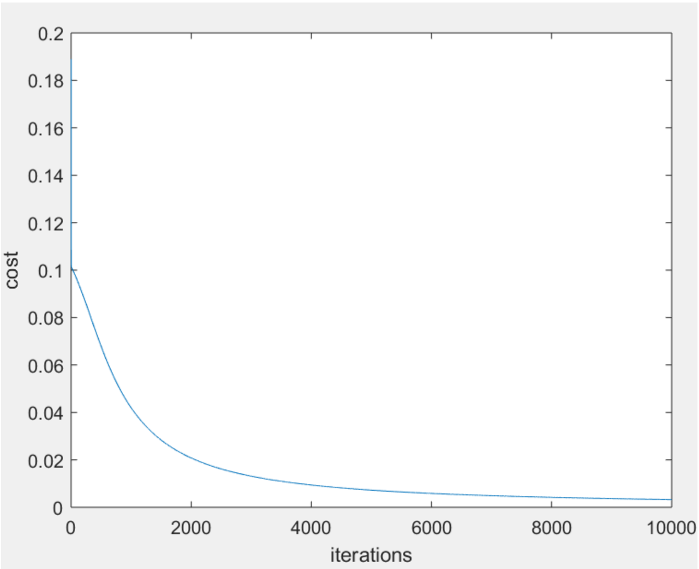


Task 11.

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.

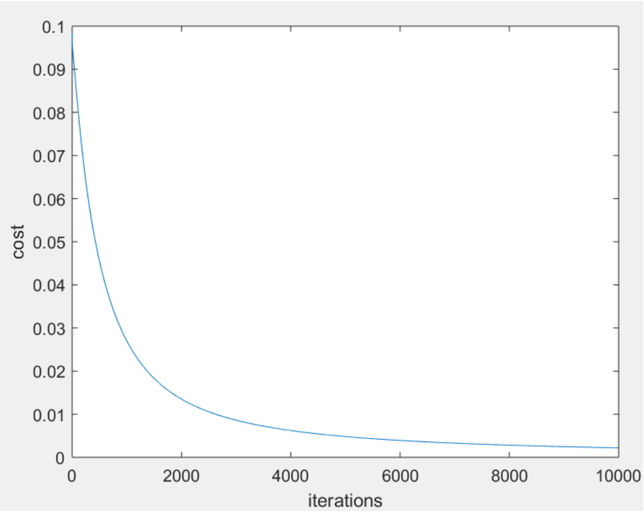
NOR

```
training_set_input = [
    0,0;
    0,1;
    1,0;
    1,1
];
```



AND

```
training_set_output = [
    0;
    0;
    0;
    1
];
```



--Thank You--

[i] www.mind.ilstu.edu/curriculum/artificial_neural_net/xor_problem_and_solution.