Machine Learning Assignment 1 (Linear Regression + Logistic Regression)

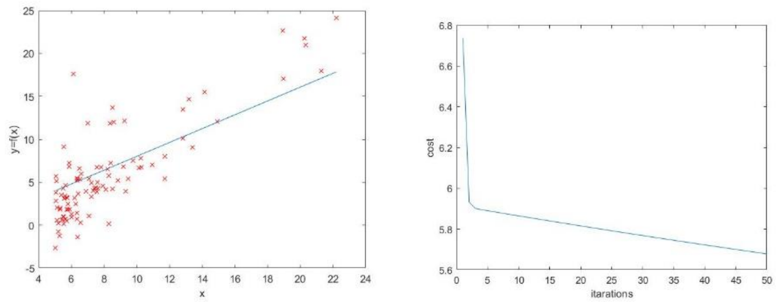
Chun Kit, Tsoi
140300468

Part 1

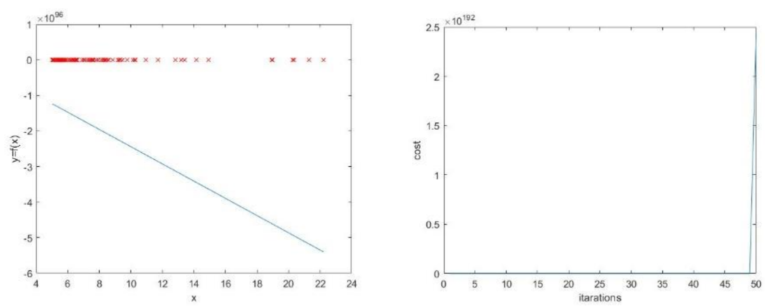
Task 1: `hypothesis = X(i,:)*theta';`

Replace by `hypothesis = calculate_hypothesis(X,theta,i);`
Left is linear regression plot. Right is cost

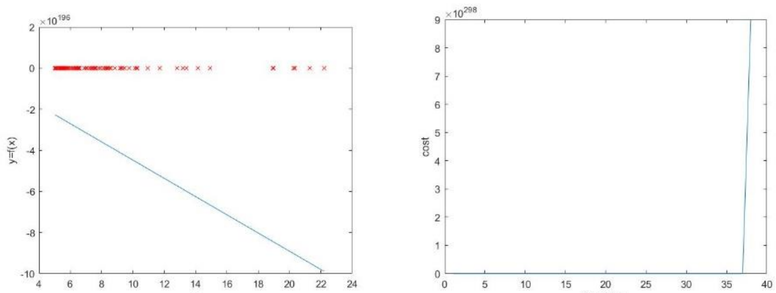
When Alpha=0.01,



When Alpha=10,



When Alpha=100,

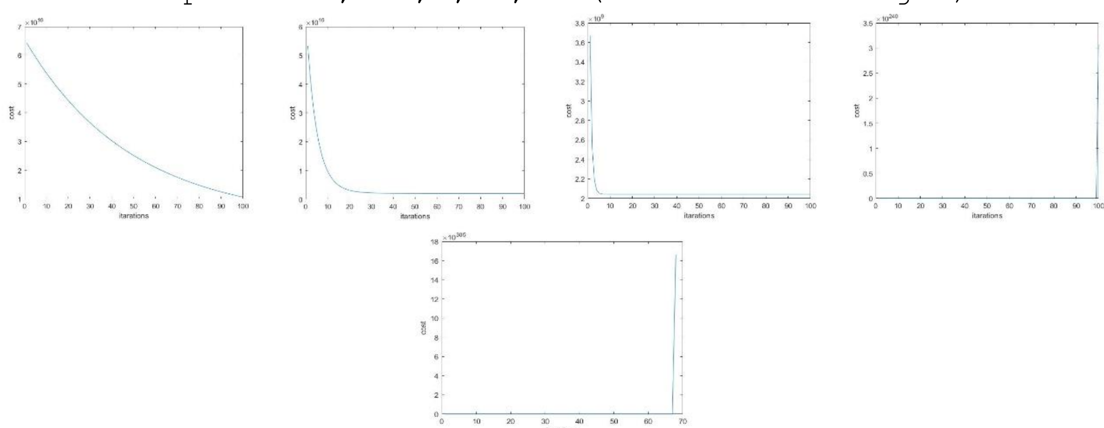


When Alpha is 0.01, the linear regression line fit well, and the cost function converge to 0 after about cost=5.7. Meanwhile, When Alpha is large, the linear regression line does not fit the data, the cost function will converge to infinity, after a long time zero-cost iteration.

Task 2:
```
hypothesis = X(i,:)*theta'; %Add to Calculate_hypothesis
```

```
%ADD to gradient_descent
theta_2 = theta(3);

        %update theta(3) and store in temporary variable theta_2
        sigma = 0.0;

        for i = 1:m
            hypothesis = calculate_hypothesis(X, theta, i);
            output = y(i);
            sigma = sigma + (hypothesis - output) * X(i, 3);
        end

         theta_2 = theta_2 - ((alpha * 1.0) / m) * sigma;

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

Alpha=0.01,0.1,1,10,100(from left to right)



Alpha=0.01 t =1.0e+05 * 2.1581 0.6138 0.2027

Alpha=0.1 t =1.0e+05 * 3.4040 1.0991 -0.0593

Alpha=1 t = 1.0e+05 * 3.4041 1.1063 -0.0665

Alpha=10 t = 1.0e+120 * -0.0000 -1.4175 -1.4175

Alpha=100 t =1.0e+222 * -0.0000 -6.4281 -6.4281

When Alpha=0.01, the cost tends to be zero. It seems to be the best learning rate.

Add `result1 = t(1) + t(2)*1650 + t(3)*3` in mllab2
result1 =1.0156e+08

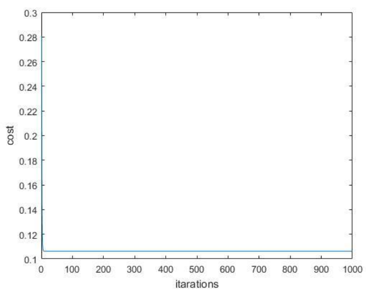
Add `result2 = t(1) + t(2)*3000 + t(3)*4` in mllab2
result2 = 1.8445e+08

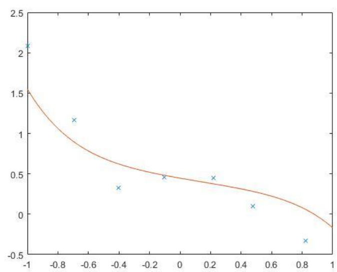
Task 3

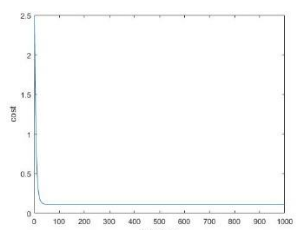
```
function theta = gradient_descent(X, y, theta, l, alpha, iterations, do_plot)
    %GRADIENT_DESCENT do Gradient Descent for a given X, y, theta, alpha
    %for a specified number of iterations

    %if less than 6 arguments was given, then set do_plot to be false
    if nargin < 6
        do_plot = false;
    end
    if(do_plot)
        plot_hypothesis(X, y, theta);
        drawnow; pause(0.1);
    end

    m = size(X, 1); %number of training examples
    num_col_theta = size(theta,2); %number of coefficients
    cost_vector = []; %will store the results of our cost function

    for it = 1:iterations
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % gradient descent
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        theta_temp = theta;

        for t = 1:num_col_theta

        sigma = 0.0;

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

        %new cost function (regularized)
        if t == 1
            theta_temp(t) = theta_temp(t) - ((alpha * 1.0) / m) * sigma;
        else
            theta_temp(t) = theta_temp(t) - ((alpha * 1.0) / m) * sigma -
theta_temp(t)*((alpha * l)/m);
        end

        end

        %update theta
        theta = theta_temp;

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

        if do_plot
```

```
            plot_hypothesis(X, y, theta);
            drawnow; pause(0.1);
        end
    end

    disp 'Gradient descent is finished.'

    if do_plot
        disp 'Press enter!'
        pause;
    end

    plot_cost(cost_vector);

    disp 'Press enter!';
    pause;
end
```

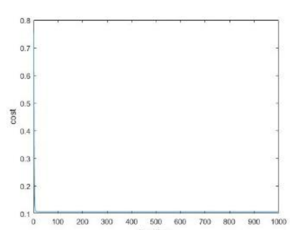
Alpha=8





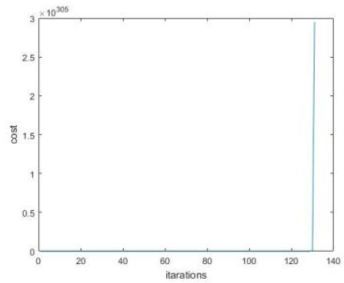


Alpha=0.1



Alpha=0.6

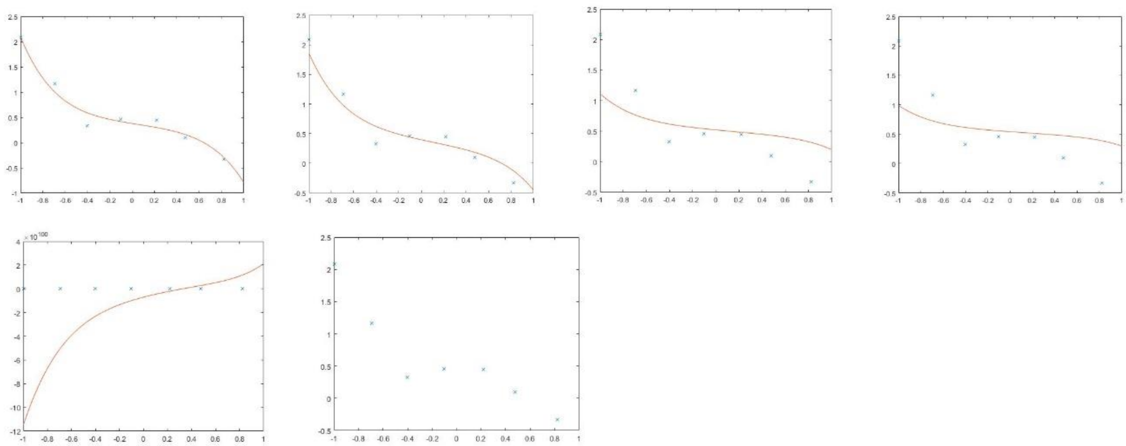
Assignment1- Chun Kit, Tsoi - 140300468



Alpha=10

The learning rate is the best at 0.6, since cost lowest.

Lamda= 0.1, 1, 10, 15, 20, 100

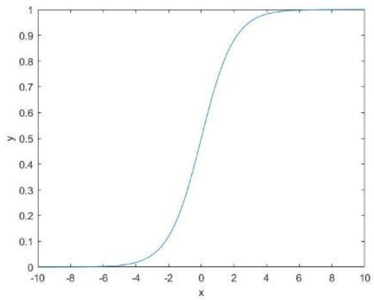


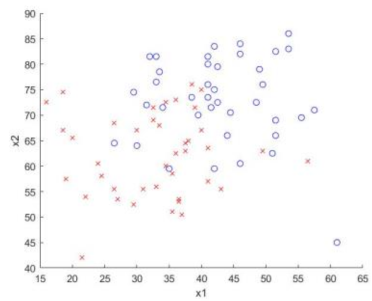
We know that if Lambda gets higher, the shape of hypothesis becomes a straight-like-line, which also causing the case of under-fit.

Part B

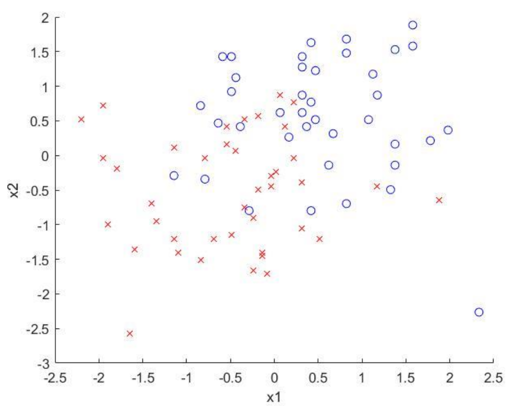
Task 1

```
function output=sigmoid(x)

    %output = 0;
    % modify this to return z passed through the sigmoid function
    %%%%%%%%%%%%%%%%%%%%%%%%
    %output= zeros(size(x));
    %output=sigmoid(x)
    output= 1.0 ./ ( 1.0 + exp(-x));

end
```





Task 2

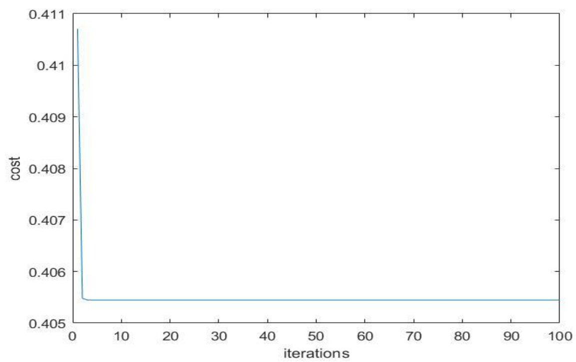


Uncomment `[X,mean,std] = normalise_features(X);`
The scale is smaller than un-normalized.

Task3

```
 hypothesis = (X(i,:)*theta');
%%%%%%%%%%%%%%%%%%%%%%%%
result=sigmoid(hypothesis);
```

Task4

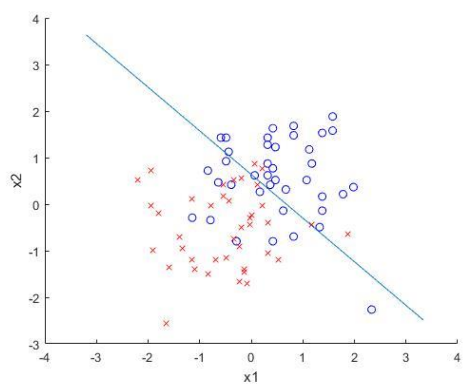



```
The final error is 0.40545.
```

Task 5

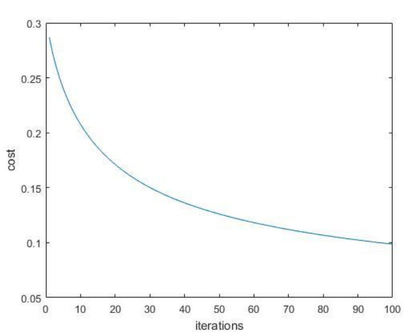
```
% modify this:
    y1 = -(theta(2)*min_x1-1 )/theta(3);
    % modify this:

    y2 = -(theta(2)*max_x1-1 )/theta(3);
```

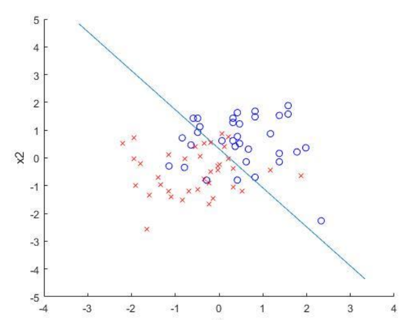
Uncomment `plot_data_function(X,y)`



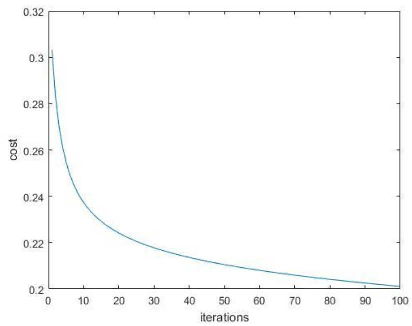
`plot_boundary(X,t);`

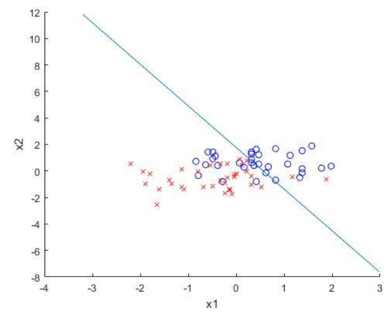
Task 6





Training error: 0.033891   Test error: 0.76183





Training error: 0.20109 Test error: 0.49358

When the training data and testing data points get close the better line and lower cost we have. Training error and test error are close to each other. As a result, the bottom graphs and better than the top graphs.

Task7

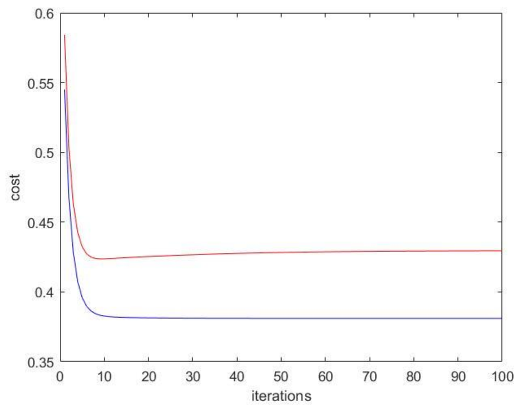
```
  for i=1:size(X:1)
         X(i,4)=X(i,2)*X(i,3);
    end
    %here append x_2 * x_2 (remember that x_1 is the bias
for i=1:size(X:1)
  X(i,5)=X(i,2)*X(i,2);

end
 %here append x_3 * x_3 (remember that x_1 is the bias
for  i=1:size(X:1)
  X(i,6)=X(i,3)*X(i,3);


end
```

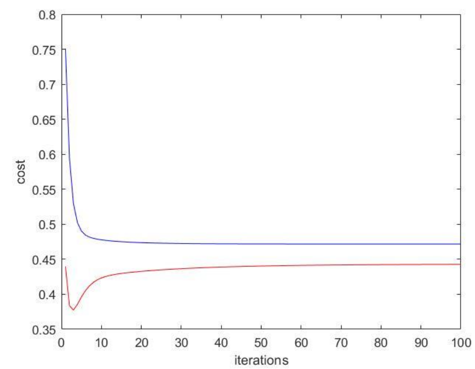
The Final error is 0.38567 lower and close the previous one. This is because we have normalized the data. We are just scaling the parameters that we have.

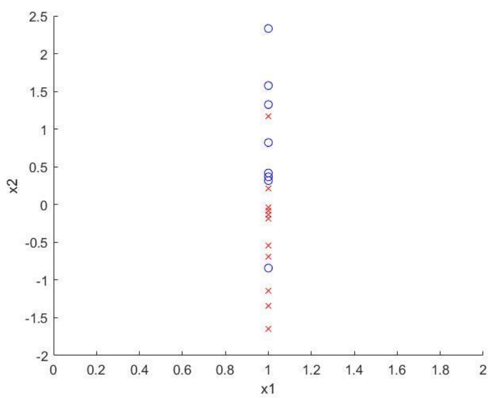
Task8

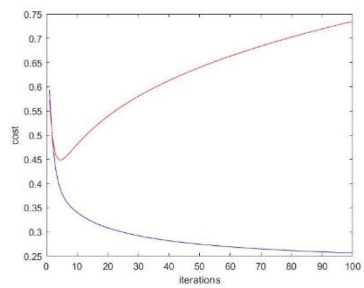
```
  for i=1:size(X:1)
         X(i,4)=X(i,2)*X(i,3);
    end
    %here append x_2 * x_2 (remember that x_1 is the bias
for i=1:size(X:1)
  X(i,5)=X(i,2)*X(i,2);

end
 %here append x_3 * x_3 (remember that x_1 is the bias
for  i=1:size(X:1)
    % update cost_array
```

```
         cost_array(it)=compute_cost(X, y, theta);
        % 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);
```









Training:0.32742

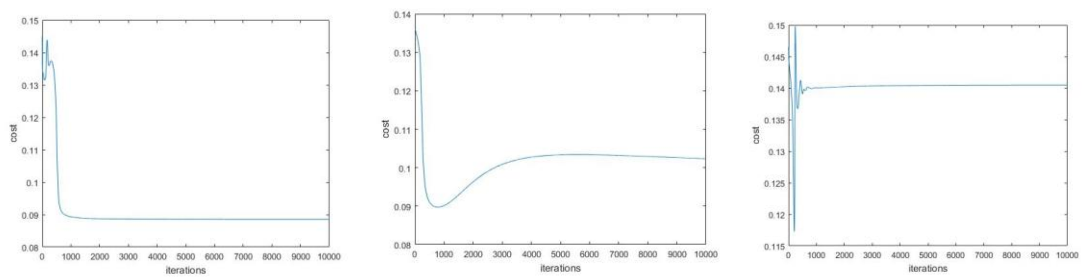
Test:0.4793

Task9

Task10

```
      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
```

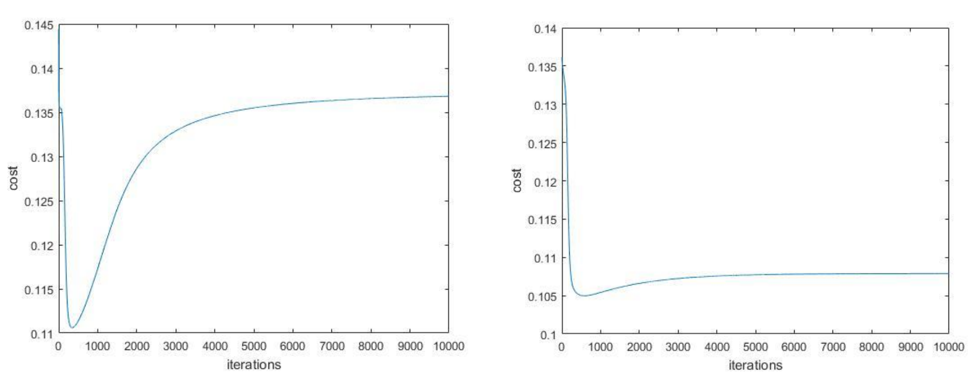
```
% Step 2. Hidden deltas (used to change weights from input --> output).
            hidden_deltas = zeros(1,length(nn.hidden_neurons));
            % hint... create a for loop here to iterate over the hidden
neurons and for each
            % hidden neuron create another for loop to iterate over the ouput
neurons
for j = 1:length(nn.hidden_neurons)
                for i = 1:length(outputs)

sum_over_outputs(j,i)=nn.hidden_weights(j,i)*output_deltas(i);
                end
hidden_deltas(j)=sigmoid_derivative(nn.hidden_neurons(j)*sum_over_outputs(j,i
));
end

   % 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
```



Learning_rate=1000, 100, 0.01

When learning rate=1000, the actual output tend to be closer to each other, 0.13327, 0.65614, 0.65834, 0.65718 and the cost is converged faster than the others which is 0.088539.

Task11



Using AND, the function is seemed to be converged faster than XOR, since we can use either conditions.

Task12

For logistic regression, we will identify 3 species of Iris by their mean, std, min, %, 50%,75% and max of sepal length, sepal width, petal length, and petal width. Then we will group their group to make a pivot table of their mean. We limit the distance between the mean and different points. Finally we draw a decision boundary of each kind. Meanwhile, Neural network runs a kind 4 times each to group them up, either yes or no happens on each node. Neural network can also solve XOR too.

Task12

iris.m not-found!!!!

There is no way I am able to do it.

Do not deduce my marks please.