**Machine Learning**

**Assignment 1**

**(Linear Regression + Logistic Regression)**

**Chun Kit, Tsoi**

**140300468**

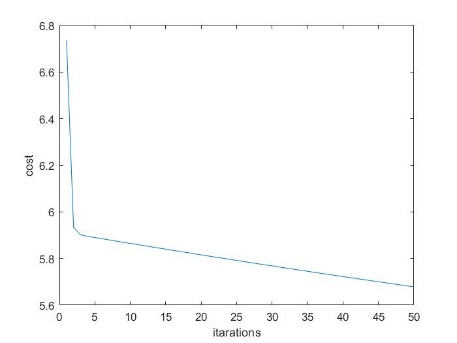
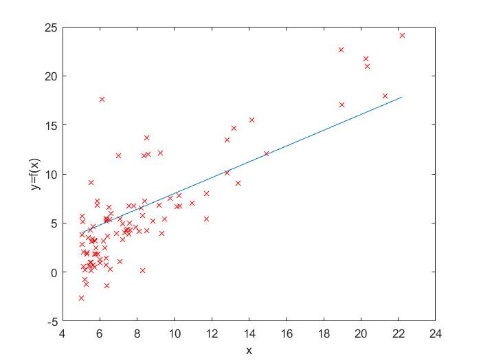
**Part 1**

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

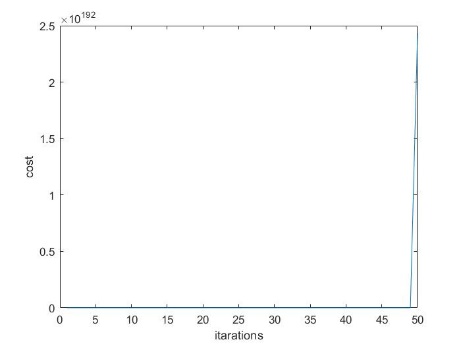
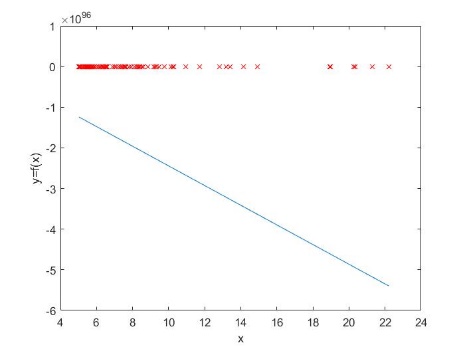
Replace by hypothesis = calculate\_hypothesis(X,theta,i);

Left is linear regression plot. Right is cost

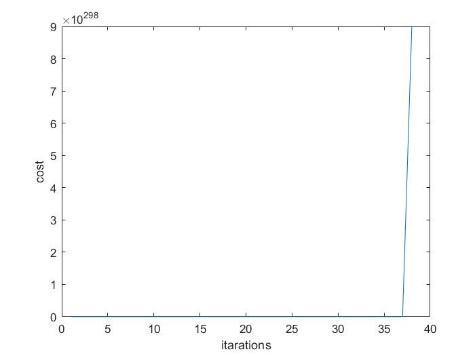
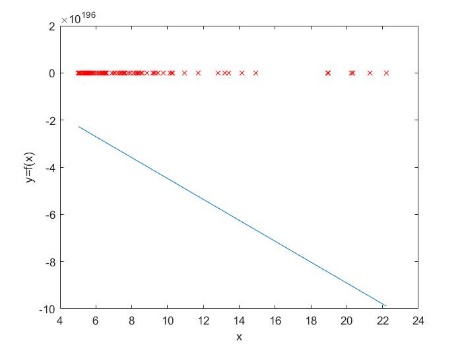
When Alpha=0.01,



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);

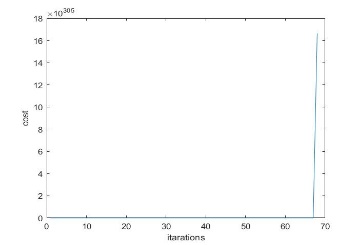
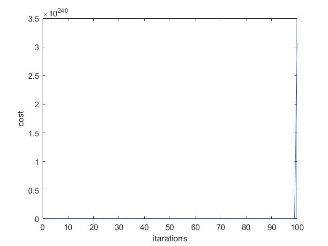
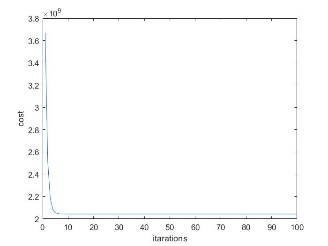
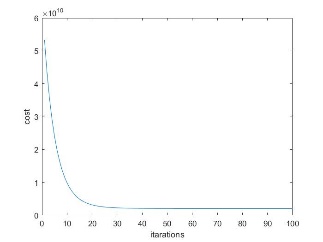
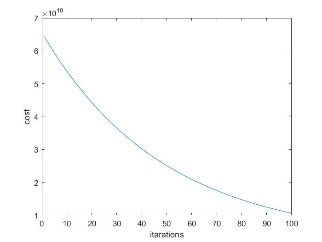
end

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

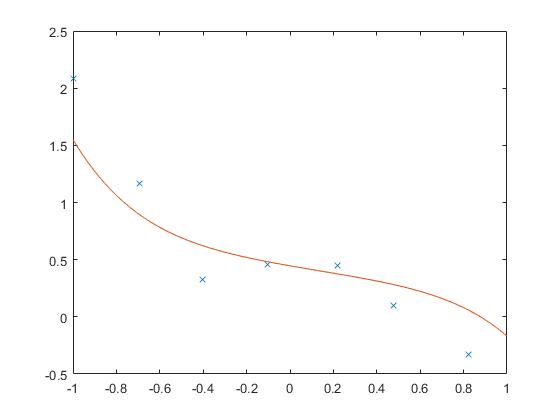
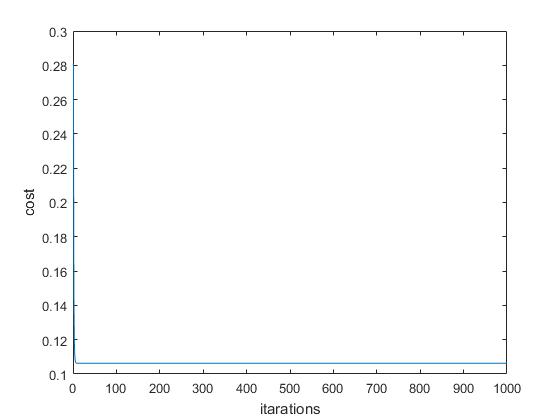
plot\_cost(cost\_vector);

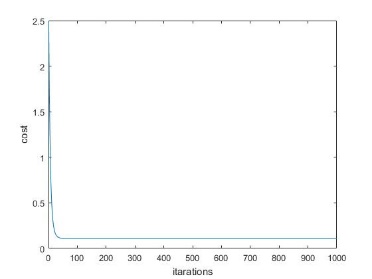
disp 'Press enter!';

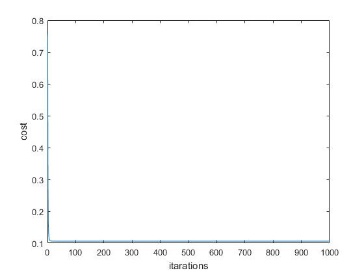
pause;

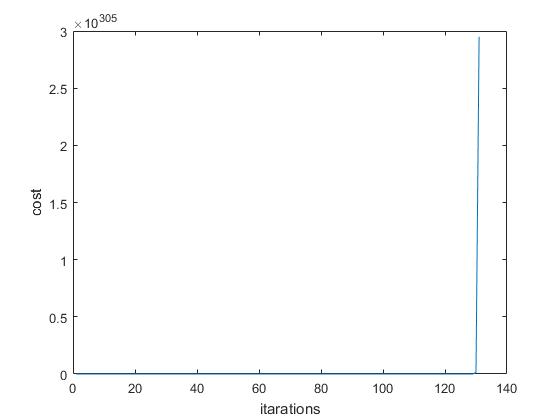
end

Alpha=8



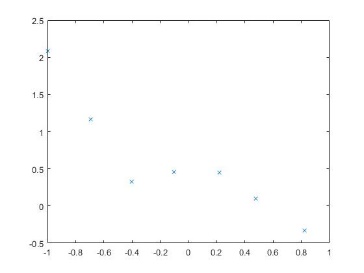
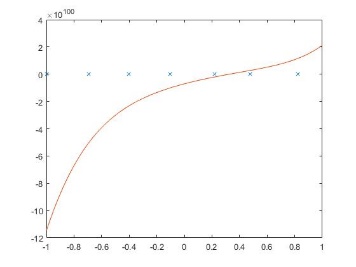
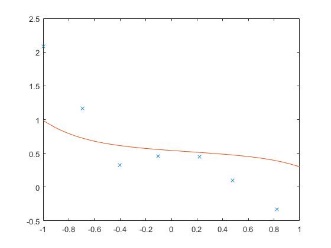
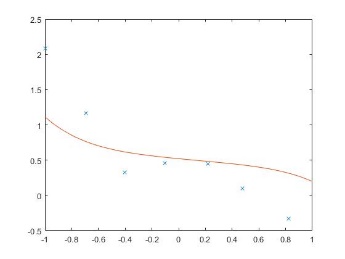
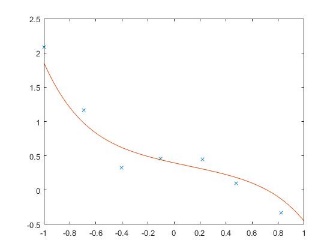
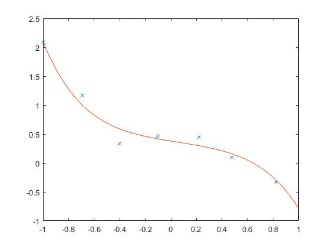
Alpha=0.1

Alpha=0.6

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 1function output=sigmoid(x)

%output = 0;

% modify this to return z passed through the sigmoid function

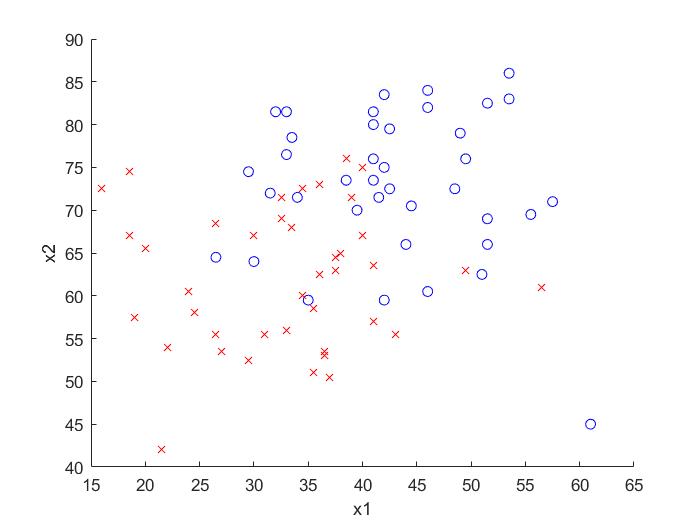
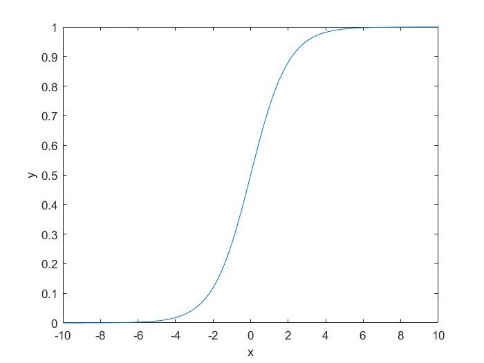
%%%%%%%%%%%%%%%%%%%%%%%%

%output= zeros(size(x));

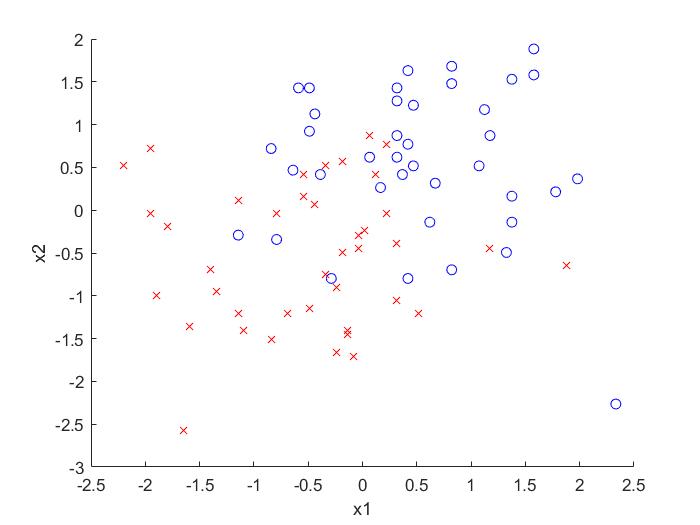
%output=sigmoid(x)

output= 1.0 ./ ( 1.0 + exp(-x));

end

****

**Task 2**

****

Uncomment [X,mean,std] = normalise\_features(X);

The scale is smaller than un-normalized.

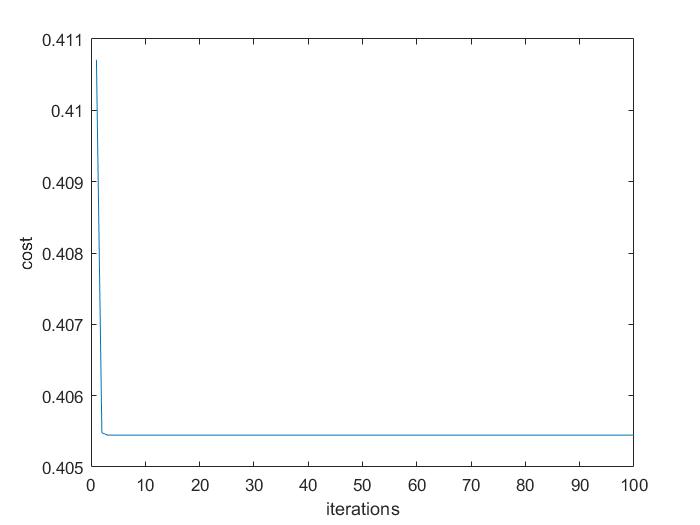
**Task3**

hypothesis = (X(i,:)\*theta');

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

result=sigmoid(hypothesis);

**Task4**



The final error is 0.40545.

**Task 5**

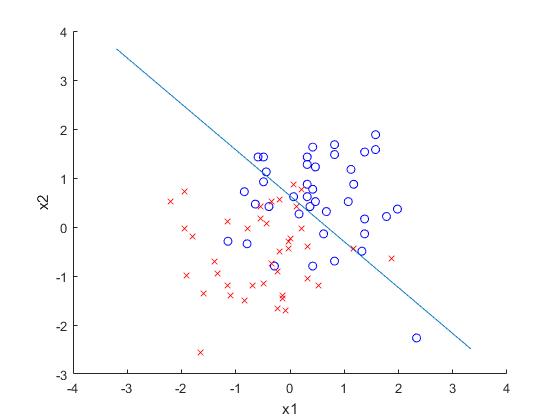
% modify this:

y1 = -(theta(2)\*min\_x1-1 )/theta(3);

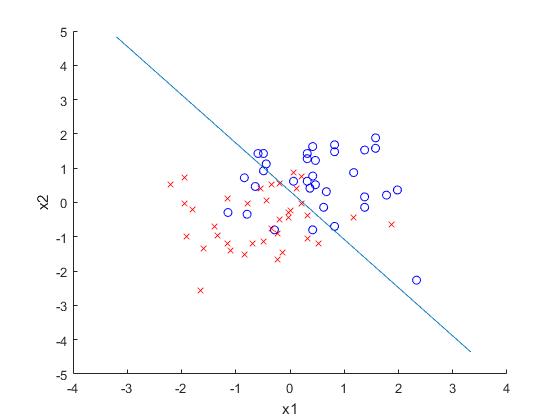
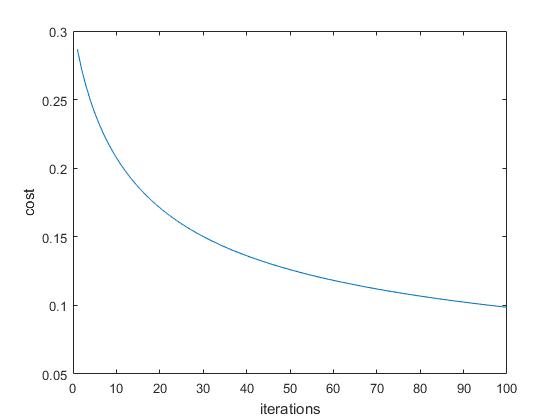
% modify this:

y2 = -(theta(2)\*max\_x1-1 )/theta(3);

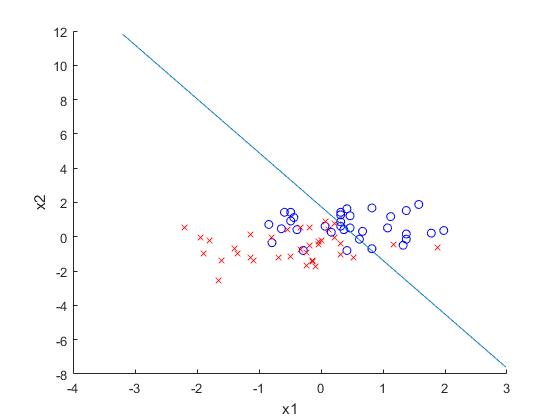
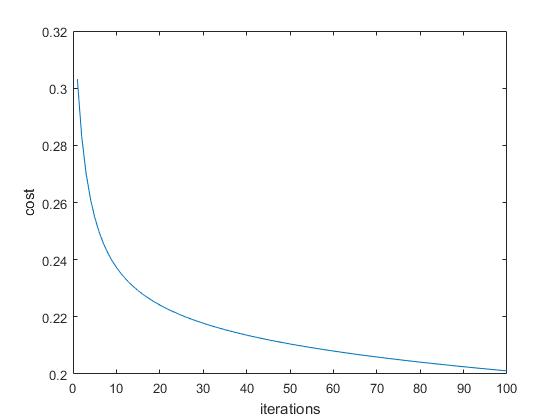
Uncomment plot\_data\_function(X,y)

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)

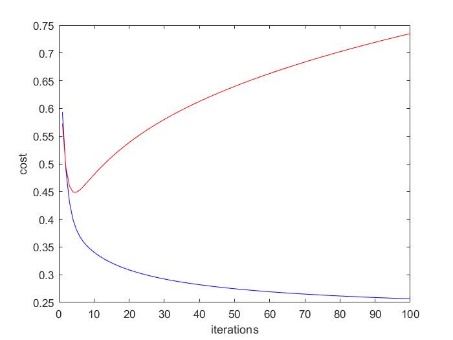
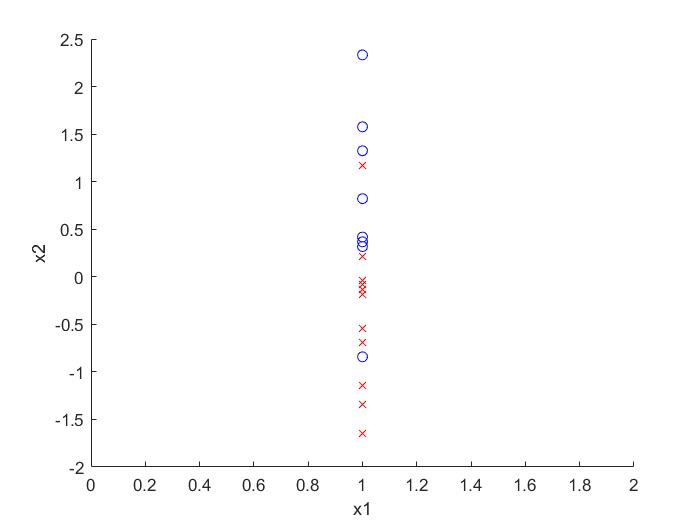
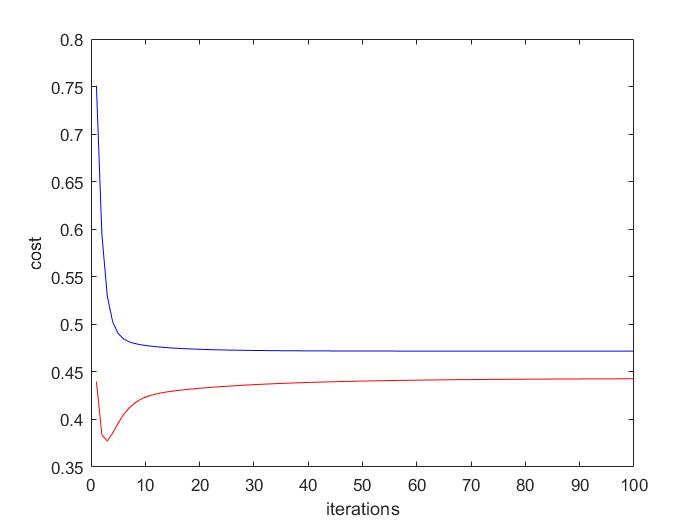
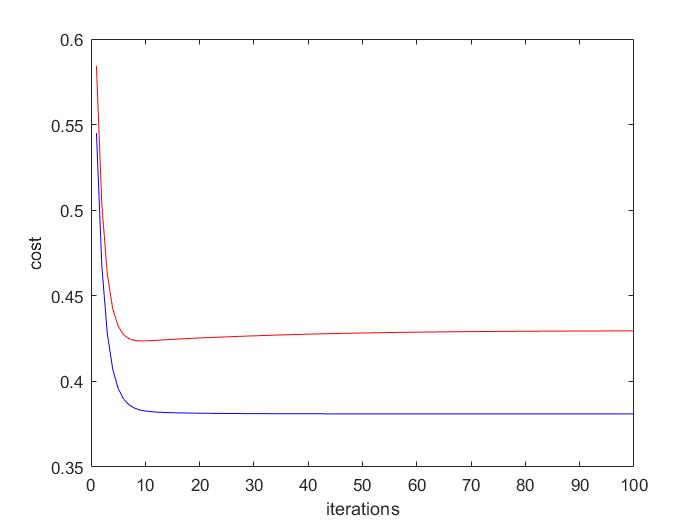
% update cost\_array

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

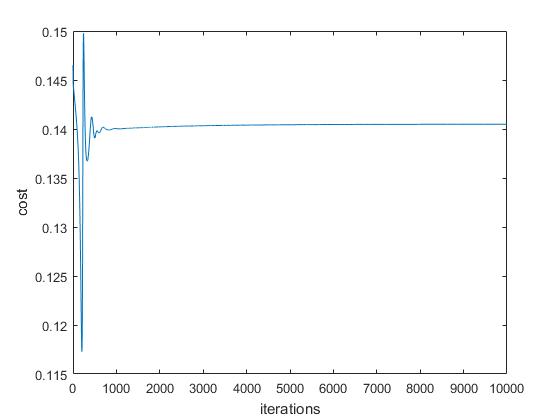
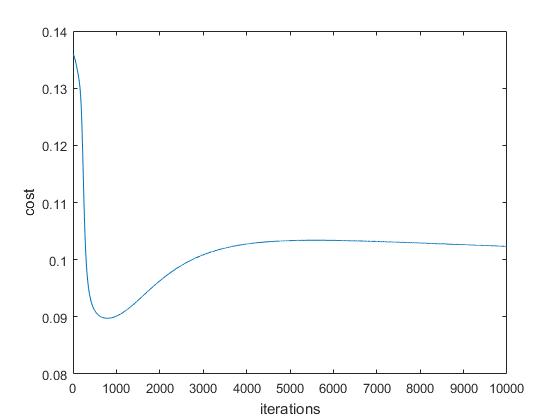
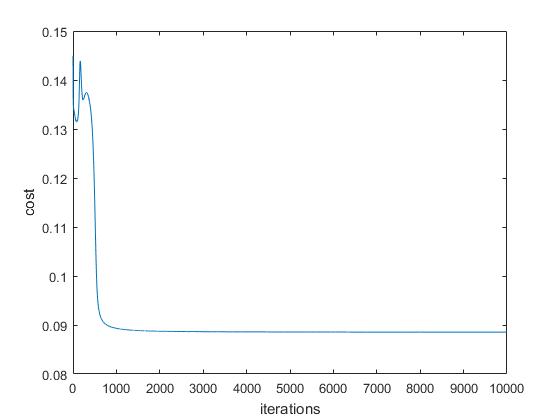
for i=1:length(nn.hidden\_neurons)

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

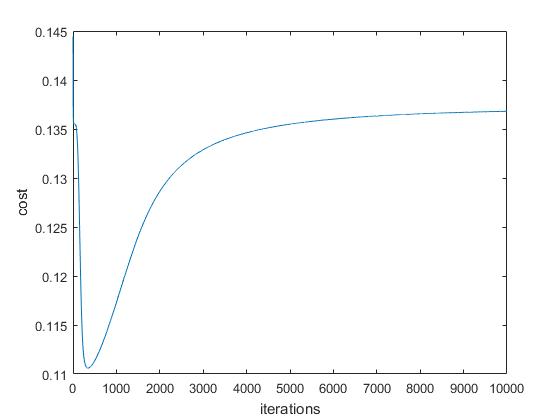
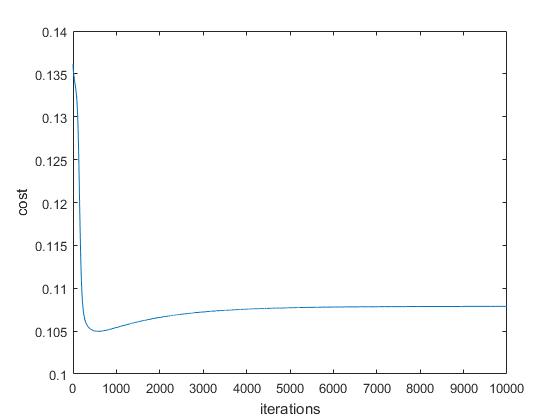
end



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.