Machina Learning Homework 1

Problem 1 solution:

Given Polynomial function
$f(x; \theta) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_d x^d$
where d is the degree of the polynomial

Given Empirical risk
$R_{emp}(\theta)=(1/N \sum_{i=1}^{N} 1/2(yi - f(x;\theta))^2$

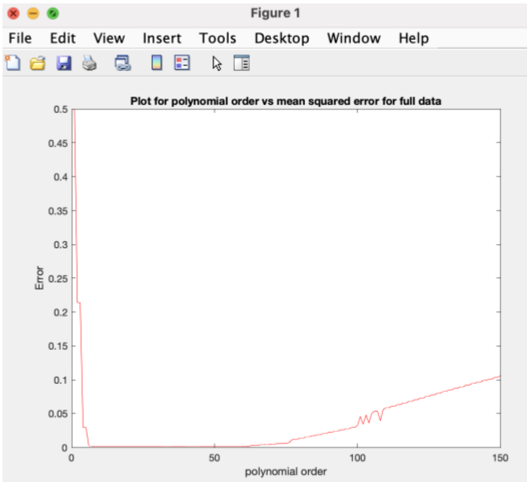
Objective is to find the polynomial degree d where the empirical risk is minimum.
Problem1.mat is having the X and Y datasets required as inputs parameters for polyreg.m function.

Matlab code:

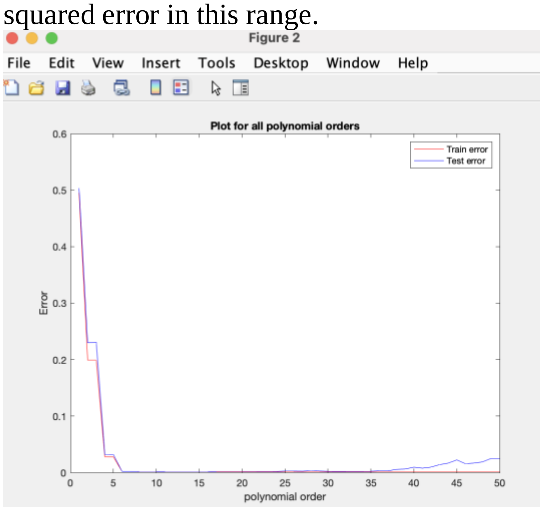
```
load problem1.mat;
Lx=length(x);
Ly=length(y);

%Shuffle the datasets x and y
ran=randperm(Lx);
x=x(ran);
y=y(ran);
x=normalize(x);
y=normalize(y);

%Fitting the data first without train and test split.
%Assuming that 100x1 would be enough
d_max=150;
error_full=zeros(d_max);
for m=1:d_max
   [err,model] = polyreg(x,y,m);
   error_full(m)=err;
end
clf
plot(error_full(1:d_max),'r');
xlabel('polynomial order');
ylabel('Error-fulldata');
title("Plot for polynomial order vs mean squared error ");

%Splitting train and test data into half's as mentioned in the problem
x_train_data=x(1:fix(Lx/2));
x_test_data=x(fix(Lx/2)+1:end);
y_train_data=y(1:fix(Lx/2));
y_test_data=y(fix(Lx/2)+1:end);

%len variable contains the maximum order of polynomial used to fit.
len= 50;
xu=1:len;
test_error=zeros(len);
```

```
train_error=zeros(len);
%fit the model for various polynomial orders
for m=1:len
   [err,model,errT] = polyreg(x_train_data,y_train_data,m,x_test_data,y_test_data);
   test_error(m)=errT;
   train_error(m)=err;
end
%plot test and train error against the polynomial order
figure()
clf
plot(train_error(1:len),'r');
hold on
plot(test_error(1:len),'b');
title("Plot for all polynomial orders");
legend('Train error','Test error');
xlabel('polynomial order');
ylabel('Error');
%The test error looks small in the 5 to 15 range
figure()
clf
plot(train_error(5:15),'r');
xticklabels({5:15})
hold on
plot(test_error(5:15),'b');
xticklabels({5:15})
title("Plot to find best polynomial order");
legend('Train error','Test error');
xlabel('polynomial order');
ylabel('Error');
%The order with lowest error and small complexity is 6
%Plot the predicted data against input data
answer=6;
[err,model,errT] = polyreg(x_train_data,y_train_data,answer,x_test_data,y_test_data);
qq = zeros(length(x),answer);
for i=1:answer
  qq(:,i) = x.^(answer-i);
end
q  = 1:500 ;
figure()
clf
scatter(x,qq*model)
hold on
scatter(x,y)
legend("predicted data","true data")
title("Plot between predicted and original data");
```

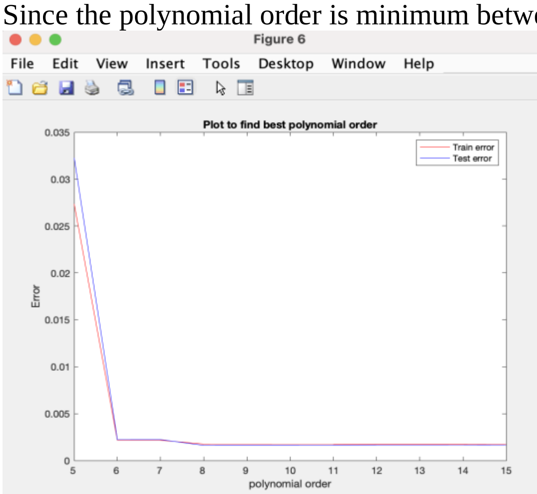
The plot for full data polynomial fit where d varies from 0 to 150



From this plot we can see that the range from 0 to 50 looks reasonable since we have the lowest mean squared error in this range.

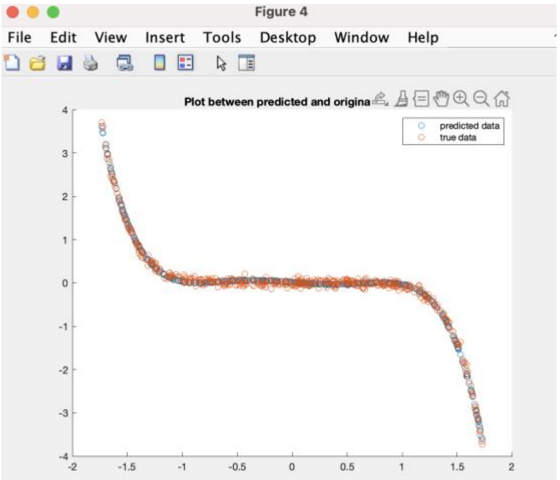


Since the polynomial order is minimum between 5 and 15 so lets look at the error between this range



The error becomes stable after **order 6** and it is best

For the polynomial order 6, comparing the predicted and input data



Conclusion: Observed that at polynomial of degree 6 is having the lowest empirical risk error

Problem 2 solution:

Given Polynomial function
$f(x; \theta) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_d x^d$
where d is the degree of the polynomial

Regularization of the Empirical risk is
$R_{emp}(\theta)=(1/N \sum_{i=1}^{N} 1/2(yi - f(x;\theta))^2 + (\lambda/2N)||\theta||^2$
Given that λ is varies from 0 to 1000

The objective is to find the λ value for which regularized empirical risk is minimum.
Problem2.mat is having the X and Y datasets required as inputs parameters for polyreg.m function.

Matlab code:

polyreg.m is rewritten as follows

```
%Including regularization factor in the model
model = inv(x'*x + lambda*eye(m))*(x'*y);

%Regularized empirical risk error
err = (1/(2*length(x)))*sum((y-x*model).^2) + (lambda/(2*length(x))) * (model'*model);

%Empirical risk error
err_unreg= (1/(2*length(x)))*sum((y-x*model).^2) ;
```

```
%Calculating test error
if (nargin==5)
errT = (1/(2*length(xT)))*sum((yT-xT*model).^2) + (lambda/(2*length(xT))) *(model'*model);
errT_unreg = (1/(2*length(xT)))*sum((yT-xT*model).^2) ;
end
```

➔ In the **homework1_2.m** matlab file

```
Lx=length(x);
Ly=length(y);

%Shuffle the datasets x and y
ran=randperm(Lx);
x=x(ran,:);
y=y(ran,:);
x=normalize(x);
y=normalize(y);

%Splitting train and test data into half's as mentioned in the problem
x_train=x(1:fix(Lx/2),:);
x_test=x(fix(Lx/2)+1:end,:);
y_train=y(1:fix(Lx/2),:);
y_test=y(fix(Lx/2)+1:end,:);
error_test=zeros(1000,1);
error_train=zeros(1000,1);
error_test_unreg=zeros(1000,1);
error_train_unreg=zeros(1000,1);

%Fitting the model for various Lambda values ranges from 0 to 1000
i = 1;
for m=0:1000
[err,err_unreg,model,errT,errT_unreg] = polyreg1_2(x_train,y_train,m,x_test,y_test);
error_test(i)=errT;
error_train(i)=err;
error_test_unreg(i)=errT_unreg;
error_train_unreg(i)=err_unreg;
i = i + 1;
end

%plot the test and train error vs Lambda value
clf
plot(error_test(1:1000),'b');
hold on
plot(error_train(1:1000),'r');
title("Regularization error vs lambda values");
legend('Test error','Train error');
xlabel('Lambda');
ylabel('Error');
figure()
clf;
plot(error_test_unreg(1:1000),'b');
```

```
hold on
plot(error_train_unreg(1:1000),'r');
title("UnRegularized error vs lambda values ");

legend('Test error','Train error');
xlabel('Lambda');
ylabel('Error');
```

Figure showing plot between regularized error and lambda values

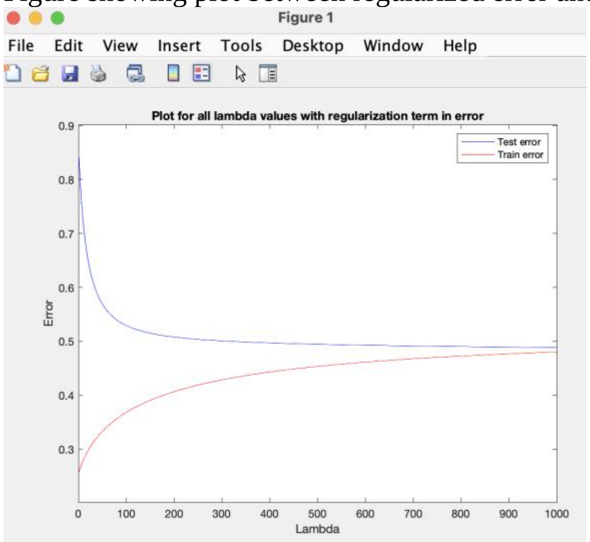
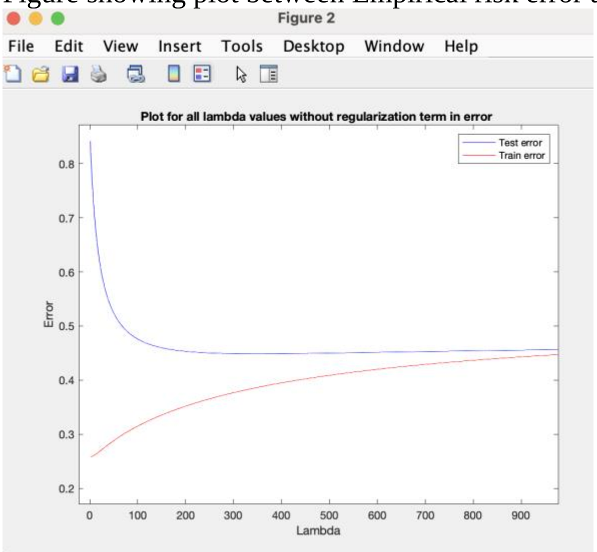


Figure showing plot between Empirical risk error and lambda values



After the lambda value around 700, error is stable at an approximate value of 0.45.

Conclusion:
Observed that lambda value is proportional to the train error and disproportionate to the test error and after a while following with a stable value.

Problem 3 solution:

Given equation is
g(z)=1/(1+exp(-z))

We have to prove that given logistic squashing function should satisfy the property
g(-z)=1-g(-z) and its inverse g^{-1}(y)=In(y/(1-y)).

g(−z) = 1/1+exp(z)

Consider 1- g(z),
1− g(z) = 1 −1/1+exp(−z)
= 1 −1/1+exp(-z)
1-g(z) =1-exp(z)/1+exp(z)
=1/1+exp(z) =g(-z)
Hence, 1-g(z) = g(-z)

For Inverse
g^{-1}(y)=x
y=g(x)
y=1/1+exp(-x)
exp(-x)=1-y/y
x=ln(y/1-y)
Hence, g^{-1}(y)=ln(y/1-y)

Problem 4 solution:

$$R_{emp}(\theta) = (\frac{1}{N})\sum_{i=1}^{n}(y_i - 1)log(1 - f(x_i;\theta) \; - y_i log(f(x_i;\theta)$$

$$\nabla_\theta R = \nabla_\theta((\frac{1}{N})\sum_{i=1}^{n}(y_i - 1)log(1 - f(x_i;\theta) - y_i log(f(x_i;\theta)))$$

$$=(\frac{1}{N})\sum_{i=1}^{n}(y_i - 1)\frac{d}{d\theta}log(1 - f(x_i;\theta)) - y_i\frac{d}{d\theta}log(f(x_i;\theta))$$

$$=(-\frac{1}{N})\sum_{i=1}^{n}(y_i\frac{1}{g(\theta^T\mathrm{x})} - (1 - y_i)\frac{1}{1-g(\theta^T\mathrm{x})})g(\theta^T\mathrm{x})(1 - g(\theta^T\mathrm{x}))\mathrm{x}$$

$$=(-\frac{1}{N})\sum_{i=1}^{n}(y - g(\theta^T\mathrm{x}))\mathrm{x}$$

We know that $\theta^{t+1} = \theta^t - \eta\nabla_\theta R_{emp}(\theta^T)$

Also for gradient descent $\theta^t - \theta^{t-1} < \varepsilon(tolerance)$
η, ε are step size and tolerance respectively.

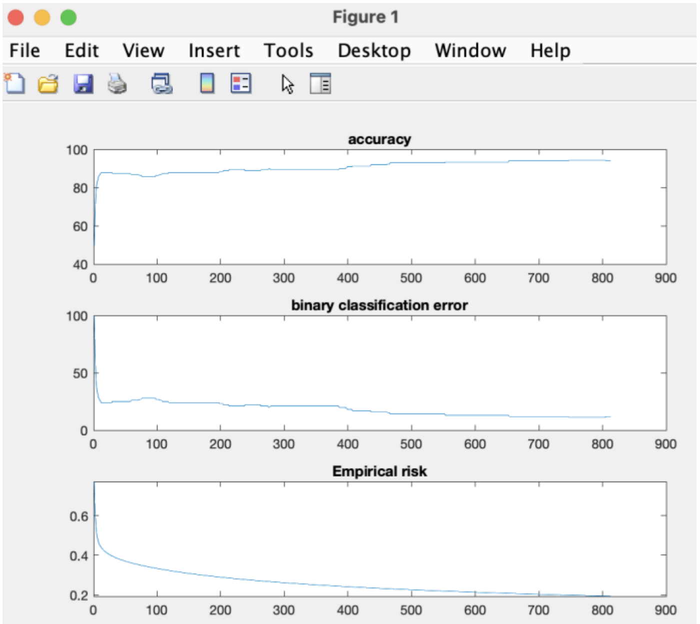
Given file dataset4.mat has the input datasets required for this problem.

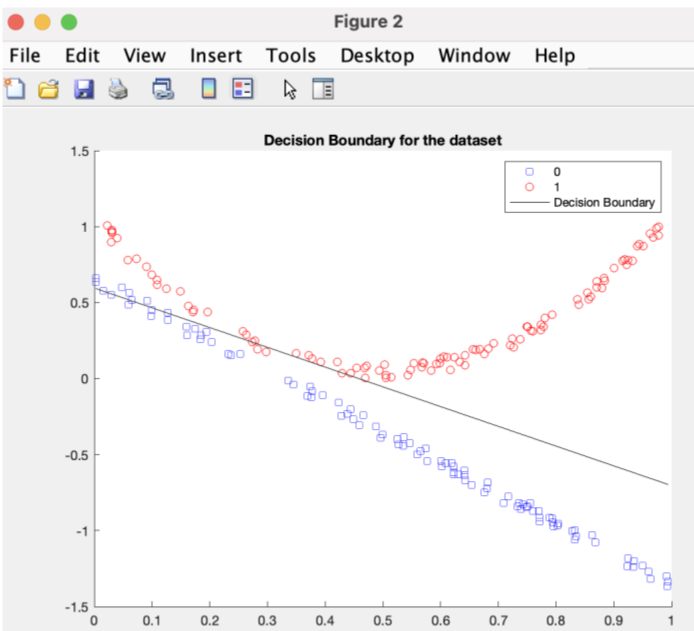
Matlab code:

```
load dataset4.mat
x_shape = size(X);

%Initialization of vectors
theta = ones(x_shape(2),1);
theta_prev = zeros(x_shape(2),1);
iter = 1;
%alpha is defined as learning rate
alpha = 0.1;
max_iter =10000;
costs = zeros(max_iter,1);
%Accuracy of prediction over test dataset
accuracy = zeros(max_iter,1);
err=zeros(max_iter,1);
tolerance = 0.001;

%When theta-theta_prev is greater than tolerance then it is overfitting
while (norm(theta-theta_prev)>tolerance) && (iter<max_iter)
    [cost,grad,f] = Remp(X,Y,theta);
    theta_prev=theta;
    theta = theta-alpha*grad;
    costs(iter) = cost;
    [acc,err_temp] = Prediction(X,Y,theta);
    accuracy(iter)=acc;
    err(iter)=err_temp;
    iter=iter+1;
end

disp("Number of iterations");
disp(iter-1);
subplot(3,1,1)
plot(1:iter-1,costs(1:iter-1))
title("Empirical risk")
subplot(3,1,2)
plot(1:iter-1,accuracy(1:iter-1))
title("accuracy")
subplot(3,1,3)
plot(1:iter-1,err(1:iter-1))
title("binary classification error")
figure()
mask1=Y==0;
mask2=Y==1;

X_out=X(mask1,:,:);
disp(X(1,1))
X_out1=X(mask2,:,:);
XX = (-theta(3)-X(:,1)*theta(1))/theta(2);

%gscatter(X(:,1),X(:,2),Y,’br’,’xo’) requires machine learning toolbox
```

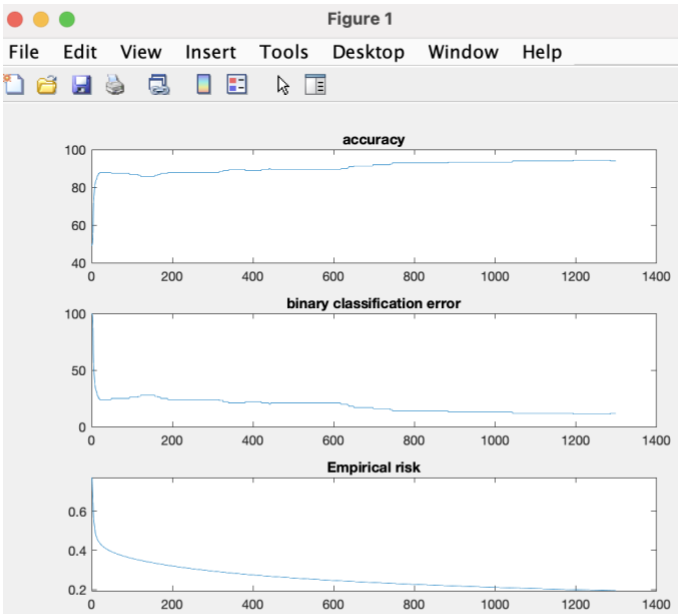
```
scatter(X_out(:,1),X_out(:,2),'bs')
hold on
scatter(X_out1(:,1),X_out1(:,2),'ro')
hold on
plot(X(:,1),XX,'k')
title("Decision Boundary for the dataset");
legend('0','1',"Decision Boundary")

%Calculating gradient and empirical risk
function [cost,grad,f] = Remp(X,Y,theta)
   m = length(Y);
   grad = zeros(size(theta));
   f = sigmoid(theta'*X')';
   cost = (-1/m)*sum(Y.*log(f)+(1-Y).*log(1-f));
   for j = 1:size(grad)
      grad(j) = (1/m)*sum((f-Y).*X(:,j));
   end
end

%f(x; θ)
function Y = sigmoid(X)
Y = 1./(1+exp(-X));
end

%Formulation of prediction function by using modified θ
function [accuracy,error] = Prediction(X,Y,theta)
   f = sigmoid(theta'*X')';
   error = 0;
   for idx = 1:size(X)
      if(f(idx)>=0.5)
         f(idx) = 1;
      end
      if(f(idx)<0.5)
         f(idx) = 0;
      end
   end
   err = Y - f;
   for idx = 1:size(X)
      if(err(idx)~=0)
         error=error+1;

      end
   end
          accuracy = 100*(size(X)-error)/size(X);
end
```

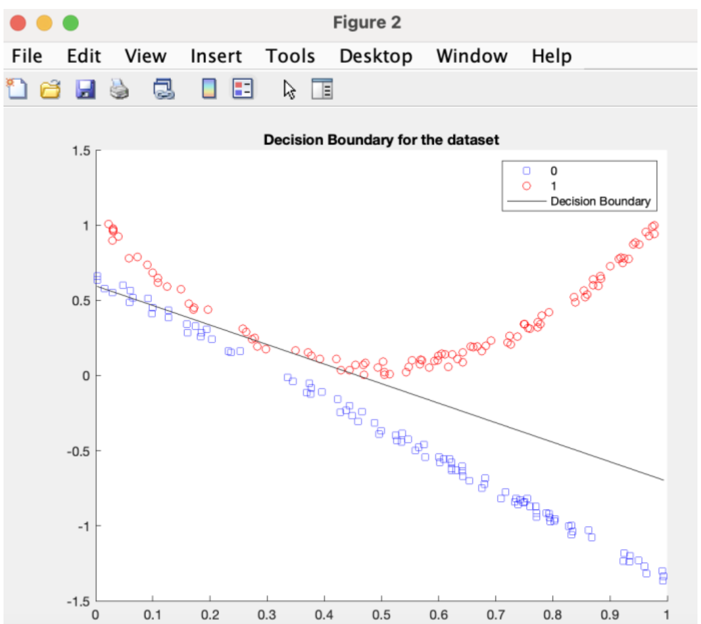
For learning rate value 0.8 and tolerance value 0.008 and with the 6500 iterations of the gradient. The figure below are the Accuracy, Binary classification error and Empirical risk.





For learning rate value 0.5 and tolerance value 0.005 and with the 6500 iterations of the gradient. The figure below are the Accuracy, Binary classification error and Empirical risk.





For learning rate value 0.1 and tolerance value 0.001 and with the 6500 iterations of the gradient. The figure below are the Accuracy, Binary classification error and Empirical risk.