**EX3 – Regularized Linear Regression and Bias v.s. Variance**

%% =========== Part 1: Loading and Visualizing Data =============

fprintf('Loading and Visualizing Data ...\n')

% You will have X, y, Xval, yval, Xtest, ytest in your environment

load ('ex5data1.mat');

m = size(X, 1);

plot(X, y, 'rx', 'MarkerSize', 10, 'LineWidth', 1.5);

xlabel('Change in water level (x)');

ylabel('Water flowing out of the dam (y)');

%% =========== Part 2: Train Linear Regression =============

lambda = 0;

[theta] = trainLinearReg([ones(m, 1) X], y, lambda);

plot(X, y, 'rx', 'MarkerSize', 10, 'LineWidth', 1.5);

xlabel('Change in water level (x)');

ylabel('Water flowing out of the dam (y)');

hold on;

plot(X, [ones(m, 1) X]\*theta, '--', 'LineWidth', 2)

hold off;

%% =========== Part 3: Learning Curve for Linear Regression =============

lambda = 0;

[error\_train, error\_val] = ...

learningCurve([ones(m, 1) X], y, ...

[ones(size(Xval, 1), 1) Xval], yval, ...

lambda);

plot(1:m, error\_train, 1:m, error\_val);

title('Learning curve for linear regression')

legend('Train', 'Cross Validation')

xlabel('Number of training examples')

ylabel('Error')

axis([0 13 0 150])

fprintf('# Training Examples\tTrain Error\tCross Validation Error\n');

for i = 1:m

fprintf(' \t%d\t\t%f\t%f\n', i, error\_train(i), error\_val(i));

end

%% =========== Part 4: Feature Mapping for Polynomial Regression ===========

p = 8;

% Map X onto Polynomial Features and Normalize

X\_poly = polyFeatures(X, p);

[X\_poly, mu, sigma] = featureNormalize(X\_poly); % Normalize

X\_poly = [ones(m, 1), X\_poly]; % Add Ones

% Map X\_poly\_test and normalize (using mu and sigma)

X\_poly\_test = polyFeatures(Xtest, p);

X\_poly\_test = bsxfun(@minus, X\_poly\_test, mu);

X\_poly\_test = bsxfun(@rdivide, X\_poly\_test, sigma);

X\_poly\_test = [ones(size(X\_poly\_test, 1), 1), X\_poly\_test];

% Map X\_poly\_val and normalize (using mu and sigma)

X\_poly\_val = polyFeatures(Xval, p);

X\_poly\_val = bsxfun(@minus, X\_poly\_val, mu);

X\_poly\_val = bsxfun(@rdivide, X\_poly\_val, sigma);

X\_poly\_val = [ones(size(X\_poly\_val, 1), 1), X\_poly\_val];

%% =========== Part 5: Learning Curve for Polynomial Regression =============

figure(2);

[error\_train, error\_val] = ...

learningCurve(X\_poly, y, X\_poly\_val, yval, lambda);

plot(1:m, error\_train, 1:m, error\_val);

title(sprintf('Polynomial Regression Learning Curve (lambda = %f)', lambda));

xlabel('Number of training examples')

ylabel('Error')

axis([0 13 0 100])

legend('Train', 'Cross Validation')

fprintf('Polynomial Regression (lambda = %f)\n\n', lambda);

fprintf('# Training Examples\tTrain Error\tCross Validation Error\n');

for i = 1:m

fprintf(' \t%d\t\t%f\t%f\n', i, error\_train(i), error\_val(i));

end

%% =========== Part 6: Validation for Selecting Lambda =============

[lambda\_vec, error\_train, error\_val] = ...

validationCurve(X\_poly, y, X\_poly\_val, yval);

close all;

plot(lambda\_vec, error\_train, lambda\_vec, error\_val);

legend('Train', 'Cross Validation');

xlabel('lambda');

ylabel('Error');

fprintf('lambda\t\tTrain Error\tValidation Error\n');

for i = 1:length(lambda\_vec)

fprintf(' %f\t%f\t%f\n', ...

lambda\_vec(i), error\_train(i), error\_val(i));

end

function [J, grad] = **linearRegCostFunction**(X, y, theta, lambda)

%LINEARREGCOSTFUNCTION Compute cost and gradient for regularized linear

%regression with multiple variables

m = length(y); % number of training examples

h\_theta = X\*theta;

Temp = theta;

Temp(1) = 0;

J = (1./(2\*m))\*sum((h\_theta-y).^2) + lambda/(2\*m)\*sum(Temp'\*Temp);

grad = (1./m)\*(X'\*(h\_theta - y)) + lambda/m \* Temp;

grad = grad(:);

end

function [theta] = **trainLinearReg**(X, y, lambda)

%TRAINLINEARREG Trains linear regression given a dataset (X, y) and a

%regularization parameter lambda

initial\_theta = zeros(size(X, 2), 1);

costFunction = @(t) linearRegCostFunction(X, y, t, lambda);

options = optimset('MaxIter', 200, 'GradObj', 'on');

theta = fmincg(costFunction, initial\_theta, options);

% theta = fminunc(costFunction, initial\_theta, options); - Also OK

end

function [error\_train, error\_val] = ...

**learningCurve**(X, y, Xval, yval, lambda)

%LEARNINGCURVE Generates the train and cross validation set errors needed

%to plot a learning curve

m = size(X, 1);

m\_cv = size(Xval,1);

error\_train = zeros(m, 1);

error\_val = zeros(m, 1);

for i=1:m

theta = trainLinearReg(X(1:i,:),y(1:i),lambda);

h\_theta = X(1:i,:)\*theta;

diff = h\_theta - y(1:i);

error\_train(i) = (1/(2\*i))\*sum(diff.^2);

h\_theta = Xval\*theta;

diff = h\_theta - yval;

error\_val(i) = (1/(2\*m\_cv))\*sum(diff.^2);

end

end

function [X\_poly] = **polyFeatures**(X, p)

%POLYFEATURES Maps X (1D vector) into the p-th power

X\_poly = zeros(numel(X), p);

for i=1:p

X\_poly(:,i) = X.^i;

end

end

function [X\_norm, mu, sigma] = **featureNormalize**(X)

%FEATURENORMALIZE Normalizes the features in X

mu = mean(X);

X\_norm = bsxfun(@minus, X, mu);

sigma = std(X\_norm);

X\_norm = bsxfun(@rdivide, X\_norm, sigma);

end

function [lambda\_vec, error\_train, error\_val] = ...

**validationCurve**(X, y, Xval, yval)

%VALIDATIONCURVE Generate the train and validation errors needed to

%plot a validation curve that we can use to select lambda

lambda\_vec = [0 0.001 0.003 0.01 0.03 0.1 0.3 1 3 10]';

error\_train = zeros(length(lambda\_vec), 1);

error\_val = zeros(length(lambda\_vec), 1);

m = size(X,1);

mVal = size(Xval,1);

for i=1:length(lambda\_vec)

theta = trainLinearReg(X,y,lambda\_vec(i));

h\_theta = X\*theta;

error\_train(i) = (1/(2\*m))\*sum((h\_theta-y).^2);

h\_theta = Xval\*theta;

error\_val(i) = (1/(2\*mVal))\*sum((h\_theta-yval).^2);

end

end

 