%% Machine Learning Online Class

% Exercise 5 | Regularized Linear Regression and Bias-Variance

%

% Instructions

% ------------

%

% This file contains code that helps you get started on the

% exercise. You will need to complete the following functions:

%

% linearRegCostFunction.m

% learningCurve.m

% validationCurve.m

%

% For this exercise, you will not need to change any code in this file,

% or any other files other than those mentioned above.

%

%% Initialization

clear ; close all; clc

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

% We start the exercise by first loading and visualizing the dataset.

% The following code will load the dataset into your environment and plot

% the data.

%

% Load Training Data

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

% Load from ex5data1:

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

load ('ex5data1.mat');

% m = Number of examples

m = size(X, 1);

% Plot training data

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

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

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

fprintf('Program paused. Press enter to continue.\n');

pause;

%% =========== Part 2: Regularized Linear Regression Cost =============

% You should now implement the cost function for regularized linear

% regression.

%

theta = [1 ; 1];

J = linearRegCostFunction([ones(m, 1) X], y, theta, 1);

fprintf(['Cost at theta = [1 ; 1]: %f '...

'\n(this value should be about 303.993192)\n'], J);

fprintf('Program paused. Press enter to continue.\n');

pause;

%% =========== Part 3: Regularized Linear Regression Gradient =============

% You should now implement the gradient for regularized linear

% regression.

%

theta = [1 ; 1];

[J, grad] = linearRegCostFunction([ones(m, 1) X], y, theta, 1);

fprintf(['Gradient at theta = [1 ; 1]: [%f; %f] '...

'\n(this value should be about [-15.303016; 598.250744])\n'], ...

grad(1), grad(2));

fprintf('Program paused. Press enter to continue.\n');

pause;

%% =========== Part 4: Train Linear Regression =============

% Once you have implemented the cost and gradient correctly, the

% trainLinearReg function will use your cost function to train

% regularized linear regression.

%

% Write Up Note: The data is non-linear, so this will not give a great

% fit.

%

% Train linear regression with lambda = 0

lambda = 0;

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

% Plot fit over the data

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;

fprintf('Program paused. Press enter to continue.\n');

pause;

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

% Next, you should implement the learningCurve function.

%

% Write Up Note: Since the model is underfitting the data, we expect to

% see a graph with "high bias" -- Figure 3 in ex5.pdf

%

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

fprintf('Program paused. Press enter to continue.\n');

pause;

%% =========== Part 6: Feature Mapping for Polynomial Regression =============

% One solution to this is to use polynomial regression. You should now

% complete polyFeatures to map each example into its powers

%

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]; % Add Ones

% 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]; % Add Ones

fprintf('Normalized Training Example 1:\n');

fprintf(' %f \n', X\_poly(1, :));

fprintf('\nProgram paused. Press enter to continue.\n');

pause;

%% =========== Part 7: Learning Curve for Polynomial Regression =============

% Now, you will get to experiment with polynomial regression with multiple

% values of lambda. The code below runs polynomial regression with

% lambda = 0. You should try running the code with different values of

% lambda to see how the fit and learning curve change.

%

lambda = 1;

[theta] = trainLinearReg(X\_poly, y, lambda);

% Plot training data and fit

figure(1);

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

plotFit(min(X), max(X), mu, sigma, theta, p);

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

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

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

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

fprintf('Program paused. Press enter to continue.\n');

pause;

%% =========== Part 8: Validation for Selecting Lambda =============

% You will now implement validationCurve to test various values of

% lambda on a validation set. You will then use this to select the

% "best" lambda value.

%

[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

fprintf('Program paused. Press enter to continue.\n');

pause;