Programming Exercise 5
Regularized Linear Regression and Bias vs Variance

Machine Learning (Stanford University)

September 2020

This exercise shows the implementation of regularized linear regression and the study of different bias-variance properties.

1 Regularized Linear Regression

Instruction:

In the first half of the exercise, you will implement regularized linear regression to predict the amount of water flowing out of a dam using the change of water level in a reservoir.

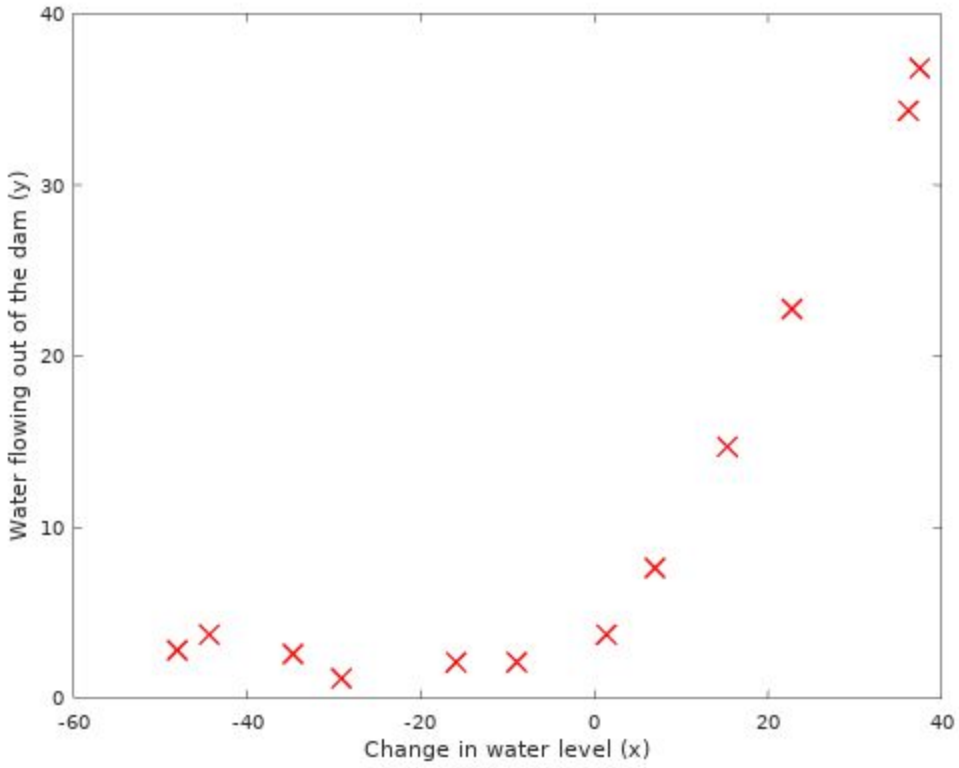
In the next half, you will go through some diagnostics of debugging learning algorithms and examine the effects of bias vs variance.

1.1 Visualizing the dataset

Dataset:
Historical records on the change in the water level, *x*, and the amount of water flowing out of the dam.

(3) Three parts

1. Training set: X,y
2. Cross Validation set for determining the regularization parameter: Xval, yval
3. Test set for evaluating performance: Xtest, test



1.2 Regularized linear regression cost function

Regularized Linear Regression's cost function:

$$J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2\right) + \frac{\lambda}{2m}\left(\sum_{j=1}^{n}\theta_j^2\right)$$

Where λ is the regularization parameter to help with overfitting

Recall:
Regularization parameter

- Controls the regularization effect
- Increasing θ_j also increases the penalty

Implementation:

linearRegCostFunction.m

```
function [J, grad] = linearRegCostFunction(X, y, theta, lambda)
%LINEARREGCOSTFUNCTION Compute cost and gradient for regularized linear
%regression with multiple variables
%   [J, grad] = LINEARREGCOSTFUNCTION(X, y, theta, lambda) computes the
%   cost of using theta as the parameter for linear regression to fit the
%   data points in X and y. Returns the cost in J and the gradient in grad

% Initialize some useful values
m = length(y); % number of training examples

% return the following variables correctly
J = 0;
grad = zeros(size(theta));

% cost function
% X: 10x3 | y: 10x1 | theta: 3x1 | lambda: 1x1 | grad: 9x1
J = (1/(2*m))*sum((X*theta - y).^2) + (lambda/(2*m))*sum(theta(2:end).^2);

grad = grad(:);
end
```

1.3 Regularized linear regression gradient

Correspondingly, the partial derivative of regularized linear regression's cost for θ_j is defined as

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad \text{for } j = 0$$

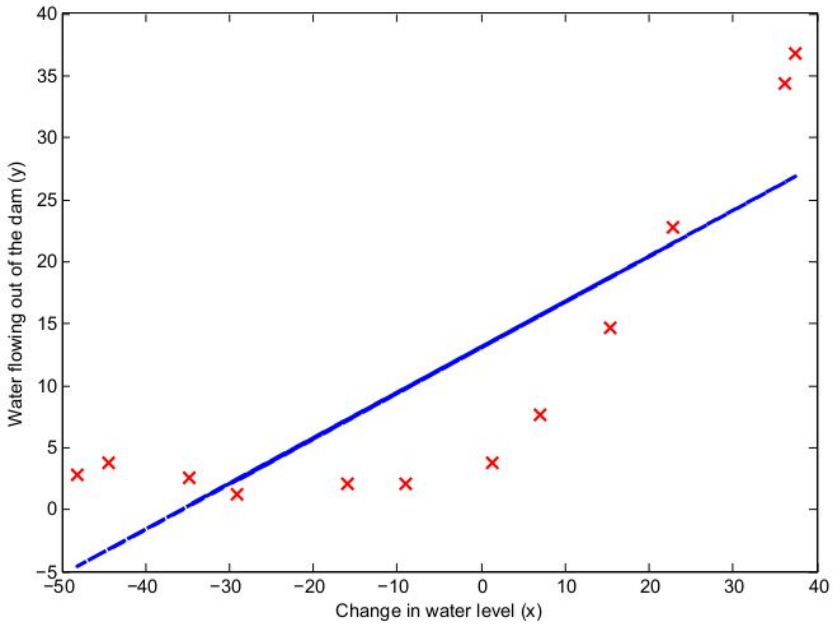
$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \qquad \text{for } j \geq 1$$

implementation:

linearRegCostFunction.m

```
function [J, grad] = linearRegCostFunction(X, y, theta, lambda)
%LINEARREGCOSTFUNCTION Compute cost and gradient for regularized linear
%regression with multiple variables
%   [J, grad] = LINEARREGCOSTFUNCTION(X, y, theta, lambda) computes the
%   cost of using theta as the parameter for linear regression to fit the
%   data points in X and y. Returns the cost in J and the gradient in grad

% Initialize some useful values
m = length(y); % number of training examples

% return the following variables correctly
J = 0;
grad = zeros(size(theta));

% cost function
% X: 10x3 | y: 10x1 | theta: 3x1 | lambda: 1x1 | grad: 9x1
J = (1/(2*m))*sum((X*theta - y).^2) + (lambda/(2*m))*sum(theta(2:end).^2);

% =========================================================================
grad(1) = ((1/m)*(X*theta- y)'*X(:,1));
grad(2:end) = (1/m)*((X*theta-y)'*X(:,2:end))' + (lambda/m)*theta(2:end);
grad = grad(:);
end
```

1.4 Fitting linear regression



Remarks:

- The best fit line tells us that the model is not a good fit to the data because the data has a non-linear pattern.
- While visualizing the best fit as shown is one possible way to debug your learning algorithm, it is not always easy to visualize the data and model.

2 Bias-variance

Remark:
An important concept in machine learning is the bias-variance tradeoff. Models with high bias are not complex enough for the data and tend to underfit, while models with high variance overfit to the training data.

Instruction:
In this part of the exercise, you will plot training and test errors on a learning curve to diagnose bias-variance problems.

2.1 Learning curves

Instruction:
You will now implement code to generate the learning curves that will be useful in debugging learning algorithms.

Recall:
learning curve

- plots training and cross validation error as a function of training set size.

Plotting the learning curve
To plot the learning curve, we need a training and cross validation set error for different training set sizes. To obtain different training set sizes, you should use different subsets of the original training set X.

Note:

- the training error

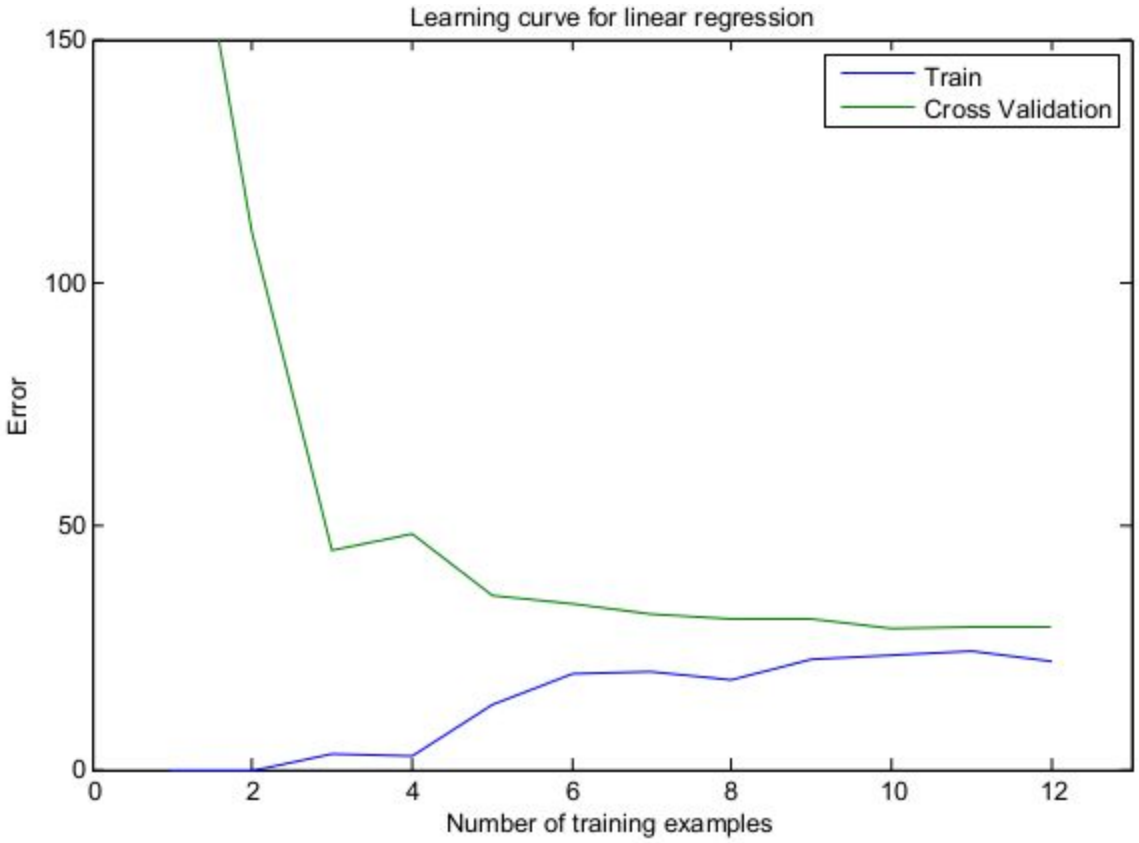
$$J_{\text{train}}(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2\right]$$

    - does not include the regularization term
        - to compute the training error is to use your existing cost function and set λ to 0 only when using it to compute the training error and cross validation error.
    - make sure you compute it on the training subset (i.e., X(1:n,:) and y(1:n)) instead of the entire training set
- for the cross validation error, you should compute it over the **entire** cross validation set.

Implementation:

learningCurve.m

```
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
%   [error_train, error_val] = ...
%       LEARNINGCURVE(X, y, Xval, yval, lambda) returns the train and
%       cross validation set errors for a learning curve. In particular,
%       it returns two vectors of the same length - error_train and
%       error_val. Then, error_train(i) contains the training error for
%       i examples (and similarly for error_val(i)).
%
%   In this function, you will compute the train and test errors for
%   dataset sizes from 1 up to m. In practice, when working with larger
%   datasets, you might want to do this in larger intervals.

% X: 10x3 | y: 10x1

m = size(X, 1); % Number of training examples

% Return these values correctly
error_train = zeros(m, 1); % 10z1
error_val   = zeros(m, 1); % 10x1

% ---------------------- Sample Solution ----------------------
for i = 1:m,
  theta = trainLinearReg(X(1:i,:), y(1:i), lambda);
  error_train(i) = linearRegCostFunction(X(1:i,:), y(1:i), theta, 0);
  error_val(i) = linearRegCostFunction(Xval, yval, theta, 0);
% -------------------------------------------------------------
end
end
```



Remark:

- We can observe that both the train error and cross validation error are high when the number of training examples is increased.

3 Polynomial regression

Remark:

- The problem with our linear model was that it was too simple for the data and resulted in underfitting (high bias).

Instruction:
In this part of the exercise, you will address this problem by adding more features.

For use polynomial regression, our hypothesis has the form:

$$h_\theta(x) = \theta_0 + \theta_1 * (\text{waterLevel}) + \theta_2 * (\text{waterLevel})^2 + \cdots + \theta_p * (\text{waterLevel})^p$$
$$= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_p x_p.$$

Notice that by defining $x_1 = (\text{waterLevel}), x_2 = (\text{waterLevel})^2, \ldots, x_p = (\text{waterLevel})^p$, we obtain a linear regression model where the features are the various powers of the original value (waterLevel).

Now, you will add more features using the higher powers of the existing feature x in the dataset. Your task in this part is to complete the code in polyFeatures.m so that the function maps the original training set X of size m × 1 into its higher powers. Specifically, when a training set X of size m × 1 is passed into the function, the function should return a m×p matrix X poly, where column 1 holds the original values of X, column 2 holds the values of X.^2, column 3 holds the values of X.^3, and so on. Note that you don't have to account for the zero-eth power in this function.

Implementation:

polyFeatures.m

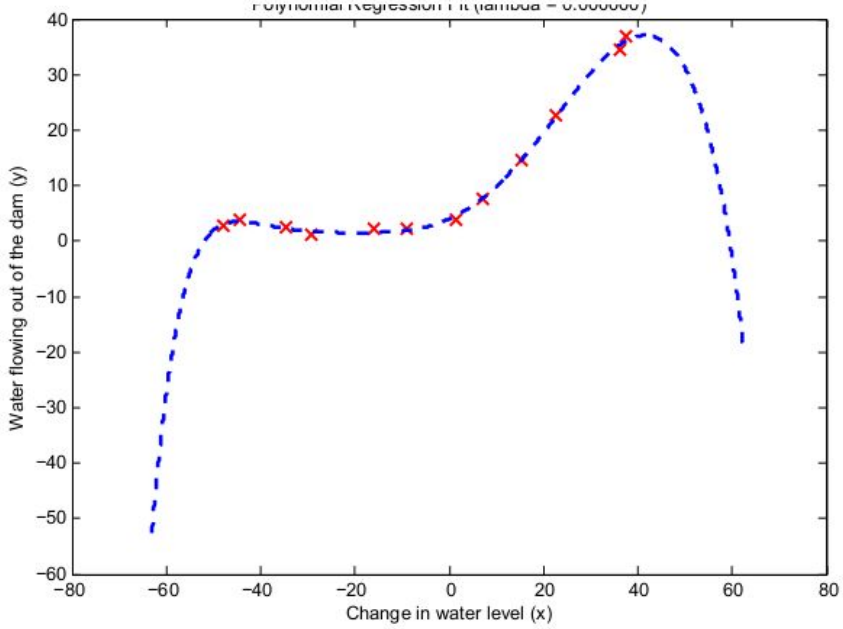
```
function [X_poly] = polyFeatures(X, p)
%POLYFEATURES Maps X (1D vector) into the p-th power
%   [X_poly] = POLYFEATURES(X, p) takes a data matrix X (size m x 1) and
%   maps each example into its polynomial features where
%   X_poly(i, :) = [X(i) X(i).^2 X(i).^3 ...  X(i).^p];
%


% You need to return the following variables correctly.
X_poly = zeros(numel(X), p);

% ====================== YOUR CODE HERE ======================
% Instructions: Given a vector X, return a matrix X_poly where the p-th
%               column of X contains the values of X to the p-th power.
%

% X: 3x1
X_poly(:,1) = X;

for i=2:p,
  X_poly(:,i) = X.^i;
% =========================================================================

end
```

3.1 Learning Polynomial Regression

Instruction:

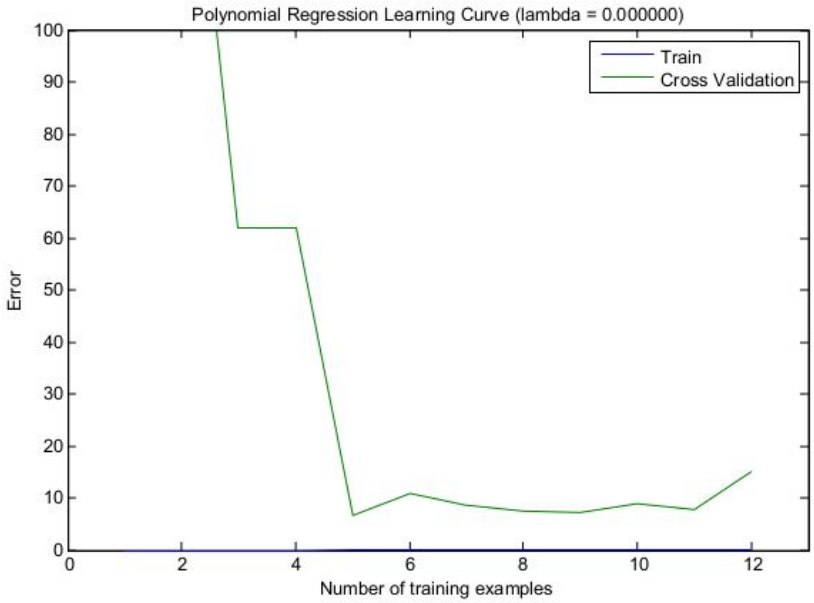
Keep in mind that even though we have polynomial terms in our feature vector, we are still solving a linear regression optimization problem. The polynomial terms have simply turned into

features that we can use for linear regression. We are using the same cost function and gradient that you wrote for the earlier part of this exercise.



Remark:

- you should see that the polynomial fit is able to follow the datapoints very well - thus, obtaining a low training error. However, the polynomial fit is very complex and even drops off at the extremes. This is an indicator that the polynomial regression model is overfitting the training data and will not generalize well.



Looking at the learning curve, this also shows the high variance problem -- the low training error is low, but the cross validation error is high.

Remark:

- One way to combat the overfitting (high-variance) problem is to add regularization to the model.

3.2 Selecting λ using a cross validation set

Remarks:

- the value of λ can significantly affect the results of regularized polynomial regression on the training and cross validation set
    - a model without regularization (λ = 0) fits the training set well, but does not generalize
    - a model with too much regularization (λ = 100) does not fit the training set and testing set well
    - A good choice of λ (e.g., λ = 1) can provide a good fit to the data.

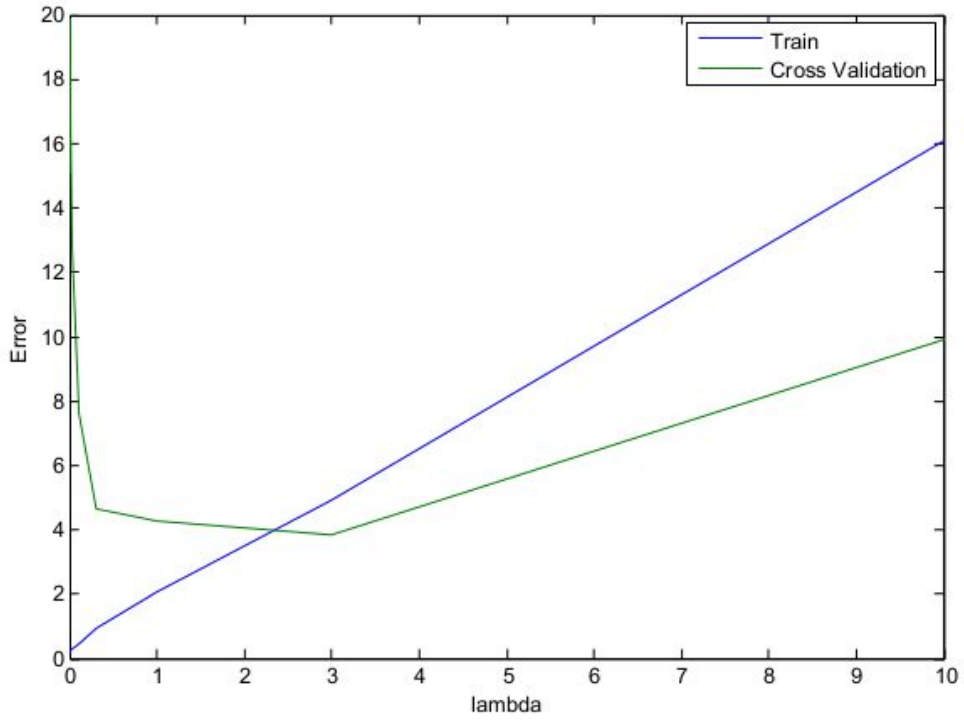
Instruction:
In this section, you will implement an automated method to select the λ parameter. Concretely, you will use a cross validation set to evaluate how good each λ value is. After selecting the best λ value using the cross validation set, we can then evaluate the model on the test set to estimate how well the model will perform on actual unseen data.

Your task is to complete the code in validationCurve.m. Specifically, you should should use the trainLinearReg function to train the model using different values of λ and compute the training error and cross validation error. You should try λ in the following range: {0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10}

Implementation:

validationCurve.m

```
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, error_train, error_val] = ...
%       VALIDATIONCURVE(X, y, Xval, yval) returns the train
%       and validation errors (in error_train, error_val)
%       for different values of lambda. You are given the training set (X,
%       y) and validation set (Xval, yval).
%

% Selected values of lambda (you should not change this)
lambda_vec = [0 0.001 0.003 0.01 0.03 0.1 0.3 1 3 10]';

% Return these variables correctly.
error_train = zeros(length(lambda_vec), 1);
error_val = zeros(length(lambda_vec), 1);

% ====================== YOUR CODE HERE ======================
% Instructions: Fill in this function to return training errors in
%               error_train and the validation errors in error_val. The
%               vector lambda_vec contains the different lambda parameters
%               to use for each calculation of the errors, i.e,
%               error train(i), and error val(i) should give
%               you the errors obtained after training with
%               lambda = lambda_vec(i)
%
% Note: You can loop over lambda_vec with the following:
%
%       for i = 1:length(lambda_vec)
%           lambda = lambda_vec(i);
%           % Compute train / val errors when training linear
%           % regression with regularization parameter lambda
%           % You should store the result in error_train(i)
%           % and error_val(i)
%           ....
%
%       end
%
%


for i = 1:length(lambda_vec),
  theta = trainLinearReg(X, y, lambda_vec(i));
  error_train(i) = linearRegCostFunction(X, y, theta, 0);
  error_val(i) = linearRegCostFunction(Xval, yval, theta, 0);
end
```



Remarks:

- In this figure, we can see that the best value of λ is around 3.
- Due to randomness in the training and validation splits of the dataset, the cross validation error can sometimes be lower than the training error.

4 Scores

```
==                                  Part Name |     Score | Feedback
==                                  --------- |     ----- | --------
== Regularized Linear Regression Cost Function |  25 /  25 | Nice work!
==      Regularized Linear Regression Gradient |  25 /  25 | Nice work!
==                              Learning Curve |  20 /  20 | Nice work!
==                  Polynomial Feature Mapping |  10 /  10 | Nice work!
==                           Validation Curve |  20 /  20 | Nice work!
==                                  --------------------------------
==                                            | 100 / 100 |
```