exercise5

June 8, 2020

1 Programming Exercise 5:

2 Regularized Linear Regression and Bias vs Variance

2.1 Introduction

In this exercise, you will implement regularized linear regression and use it to study models with different bias-variance properties. Before starting on the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

All the information you need for solving this assignment is in this notebook, and all the code you will be implementing will take place within this notebook. The assignment can be promptly submitted to the coursera grader directly from this notebook (code and instructions are included below).

Before we begin with the exercises, we need to import all libraries required for this programming exercise. Throughout the course, we will be using `numpy` for all arrays and matrix operations, `matplotlib` for plotting, and `scipy` for scientific and numerical computation functions and tools. You can find instructions on how to install required libraries in the README file in the github repository.

In [1]:
```
# used for manipulating directory paths
import os

# Scientific and vector computation for python
import numpy as np

# Plotting library
from matplotlib import pyplot

# Optimization module in scipy
from scipy import optimize

# will be used to load MATLAB mat datafile format
from scipy.io import loadmat

# library written for this exercise providing additional functions for assignment subm
import utils
```

```
# define the submission/grader object for this exercise
grader = utils.Grader()

# tells matplotlib to embed plots within the notebook
%matplotlib inline
```

2.2 Submission and Grading

After completing each part of the assignment, be sure to submit your solutions to the grader. The following is a breakdown of how each part of this exercise is scored.

	Section	Part	Submitted Function	Points
1		Section ??	Section ??	25
2		Section ??	Section ??	25
3		Section ??	Section ??	20
4		Section ??	Section ??	10
5		Section ??	Section ??	20
		Total Points		100

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration.

At the end of each section in this notebook, we have a cell which contains code for submitting the solutions thus far to the grader. Execute the cell to see your score up to the current section. For all your work to be submitted properly, you must execute those cells at least once.

1 Regularized Linear Regression

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 v.s. variance.

2.2.1 1.1 Visualizing the dataset

We will begin by visualizing the dataset containing historical records on the change in the water level, x, and the amount of water flowing out of the dam, y. This dataset is divided into three parts:

- A **training** set that your model will learn on: `X`, `y`
- A **cross validation** set for determining the regularization parameter: `Xval`, `yval`
- A **test** set for evaluating performance. These are "unseen" examples which your model did not see during training: `Xtest`, `ytest`

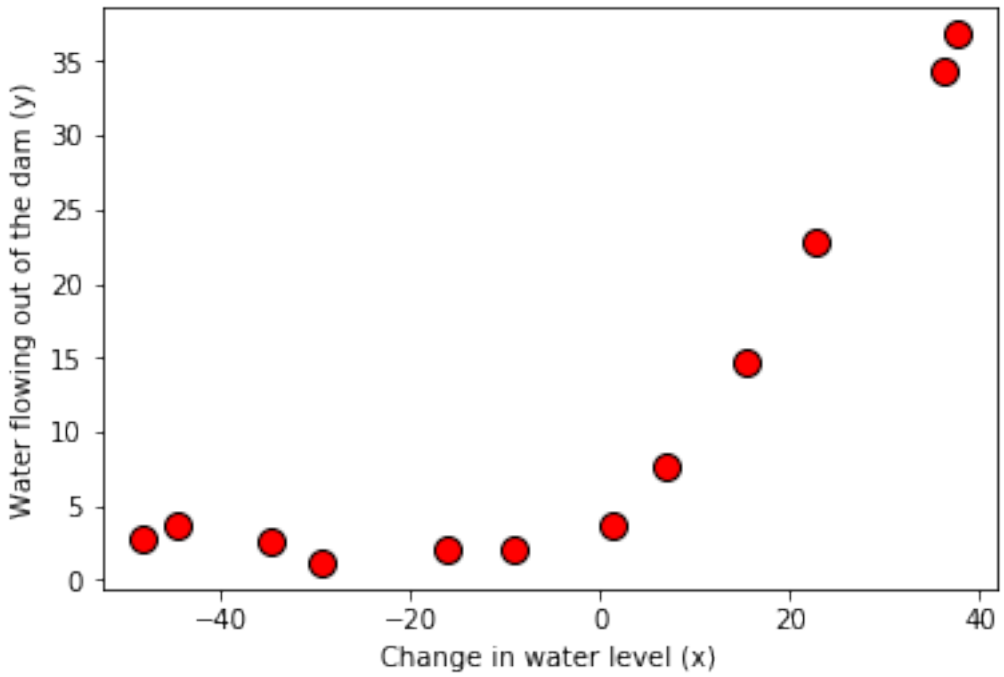
Run the next cell to plot the training data. In the following parts, you will implement linear regression and use that to fit a straight line to the data and plot learning curves. Following that, you will implement polynomial regression to find a better fit to the data.

In [2]:
```
# Load from ex5data1.mat, where all variables will be store in a dictionary
data = loadmat(os.path.join('Data', 'ex5data1.mat'))
```

```
# Extract train, test, validation data from dictionary
# and also convert y's form 2-D matrix (MATLAB format) to a numpy vector
X, y = data['X'], data['y'][:, 0]
Xtest, ytest = data['Xtest'], data['ytest'][:, 0]
Xval, yval = data['Xval'], data['yval'][:, 0]

# m = Number of examples
m = y.size

# Plot training data
pyplot.plot(X, y, 'ro', ms=10, mec='k', mew=1)
pyplot.xlabel('Change in water level (x)')
pyplot.ylabel('Water flowing out of the dam (y)');
```



2.2.2 1.2 Regularized linear regression cost function

Recall that regularized linear regression has the following cost function:

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

where λ is a regularization parameter which controls the degree of regularization (thus, help preventing overfitting). The regularization term puts a penalty on the overall cost J. As the magnitudes of the model parameters θ_j increase, the penalty increases as well. Note that you should not regularize the θ_0 term.

You should now complete the code in the function `linearRegCostFunction` in the next cell. Your task is to calculate the regularized linear regression cost function. If possible, try to vectorize your code and avoid writing loops.

```
In [3]: def linearRegCostFunction(X, y, theta, lambda_=0.0):
            """
            Compute cost and gradient for regularized linear regression
            with multiple variables. Computes the cost of using theta as
            the parameter for linear regression to fit the data points in X and y.

            Parameters
            ----------
            X : array_like
                The dataset. Matrix with shape (m x n + 1) where m is the
                total number of examples, and n is the number of features
                before adding the bias term.

            y : array_like
                The functions values at each datapoint. A vector of
                shape (m, ).

            theta : array_like
                The parameters for linear regression. A vector of shape (n+1,).

            lambda_ : float, optional
                The regularization parameter.

            Returns
            -------
            J : float
                The computed cost function.

            grad : array_like
                The value of the cost function gradient w.r.t theta.
                A vector of shape (n+1, ).

            Instructions
            ------------
            Compute the cost and gradient of regularized linear regression for
            a particular choice of theta.
            You should set J to the cost and grad to the gradient.
            """
            # Initialize some useful values
            m = y.size # number of training examples

            # You need to return the following variables correctly
            J = 0
            grad = np.zeros(theta.shape)
```

```
            # ====================== YOUR CODE HERE ======================

            h = X.dot(theta)

            J = (1 / (2*m)) * np.sum(np.square(h-y)) + (lambda_ / (2*m)) * np.sum(np.square(the

            grad = (1 / m) * (h - y).dot(X)

            grad[1:] = grad[1:] + (lambda_ / m) * theta[1:]

            # ============================================================
            return J, grad
```

When you are finished, the next cell will run your cost function using `theta` initialized at `[1, 1]`. You should expect to see an output of 303.993.

```
In [4]: theta = np.array([1, 1])
        J, _ = linearRegCostFunction(np.concatenate([np.ones((m, 1)), X], axis=1), y, theta, 1)

        print('Cost at theta = [1, 1]:\t   %f ' % J)
        print('This value should be about 303.993192)\n' % J)
```

```
Cost at theta = [1, 1]:           303.993192
This value should be about 303.993192)
```

After completing a part of the exercise, you can submit your solutions for grading by first adding the function you modified to the submission object, and then sending your function to Coursera for grading.

The submission script will prompt you for your login e-mail and submission token. You can obtain a submission token from the web page for the assignment. You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration.

Execute the following cell to grade your solution to the first part of this exercise.

```
In [6]: grader[1] = linearRegCostFunction
        grader.grade()
```

```
Submitting Solutions | Programming Exercise regularized-linear-regression-and-bias-variance

Use token from last successful submission (saurabh.takle@gmail.com)? (Y/n): y
                                  Part Name |     Score | Feedback
                                  --------- |     ----- | --------
Regularized Linear Regression Cost Function |  25 /  25 | Nice work!
     Regularized Linear Regression Gradient |   0 /  25 |
                             Learning Curve |   0 /  20 |
                 Polynomial Feature Mapping |   0 /  10 |
```