Programming Exercise 1: Linear Regression

Machine Learning

1 Introduction

In this exercise, you will implement linear regression and get to see it work on data. Before starting on this programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise. If needed, use the *setwd()* command in R to change to this directory before starting this exercise or open the project Ex1.Rproj in R Studio

1.1 Files included in this exercise

- ex1.R - R script that will help step you through the exercise
- ex1_multi.R - R script for the later parts of the exercise
- ex1data1.txt - Dataset for linear regression with one variable
- ex1data2.txt - Dataset for linear regression with multiple variables
- submit.R - Submission script that sends your solutions to our servers
- [⋆] warmUpExercise.R - Simple example function in R
- [⋆] plotData.R - Function to display the dataset
- [⋆] computeCost.R - Function to compute the cost of linear regression
- [⋆] gradientDescent.R - Function to run gradient descent
- [†] computeCostMulti.R - Cost function for multiple variables
- [†] gradientDescentMulti.R - Gradient descent for multiple variables
- [†] featureNormalize.R - Function to normalize features
- [†] normalEqn.R - Function to compute the normal equations

⋆ indicates you will need to complete.
† indicates extra credit exercises.

Throughout the exercise, you will be using the scripts *ex1.R* and *ex1_multi.R*.

These scripts set up the dataset for the problems and make calls to functions that you will write. You do not need to modify either of them. You are only required to modify functions in other files, by following the instructions in this assignment.

For this programming exercise, you are only required to complete the first part of the exercise to implement linear regression with one variable. The second part of the exercise, which you may complete for extra credit, covers linear regression with multiple variables.

1.2 Where to get help

The exercises in this course use R

2 Simple R function

The first part of *ex1.R* gives you practice with R syntax and the homework submission process. In the file warmUpExercise.R, you will find the outline of an R function. Modify it to return a 5 x 5 identity matrix by filling in the following code:

```
A = diag(5);
```

When you are finished, run ex1.R (assuming you are in the correct directory, type *source("ex1.R")* at the R prompt) and you should see output similar to the following:

```
## [1] "Running warmUpExercise ..."

## [1] "5x5 Identity Matrix: "

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
```

Now ex1.R will pause until you press any key, and then will run the code for the next part of the assignment. If you wish to quit, typing ctrl-c will stop the program in the middle of its run.

2.1 Submitting Solutions

After completing a part of the exercise, you can submit your solutions for grading by typing submit at the R command line. The submission script will prompt you for your username and password and ask you which files you want to submit. You can obtain a submission password from the website's "Programming Exercises" page.

You should now submit the warm up exercise.

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration. To prevent rapid-fire guessing, the system enforces a minimum of 5 minutes between submissions.

---

3 Linear regression with one variable

In this part of this exercise, you will implement linear regression with one variable to predict profits for a food truck. Suppose you are the CEO of a restaurant franchise and are considering diferent cities for opening a new outlet. The chain already has trucks in various cities and you have data for profits and populations from the cities.
You would like to use this data to help you select which city to expand to next.

The file *ex1data1.txt* contains the dataset for our linear regression problem. The first column is the population of a city and the second column is the profit of a food truck in that city. A negative value for profit indicates a loss.
The *ex1.R* script has already been set up to load this data for you.

3.1 Plotting the Data

Before starting on any task, it is often useful to understand the data by visualizing it. For this dataset, you can use a scatter plot to visualize the data, since it has only two properties to plot (profit and population). (Many other problems that you will encounter in real life are multi-dimensional and can't be plotted on a 2-d plot.)

In *ex1.m*, the dataset is loaded from the data file into the variables X and y:

```
data <- read.csv('ex1data1.txt', header=FALSE);
X <- data[, 1]; y <- data[, 2];
m <- length(y); # number of training examples
```

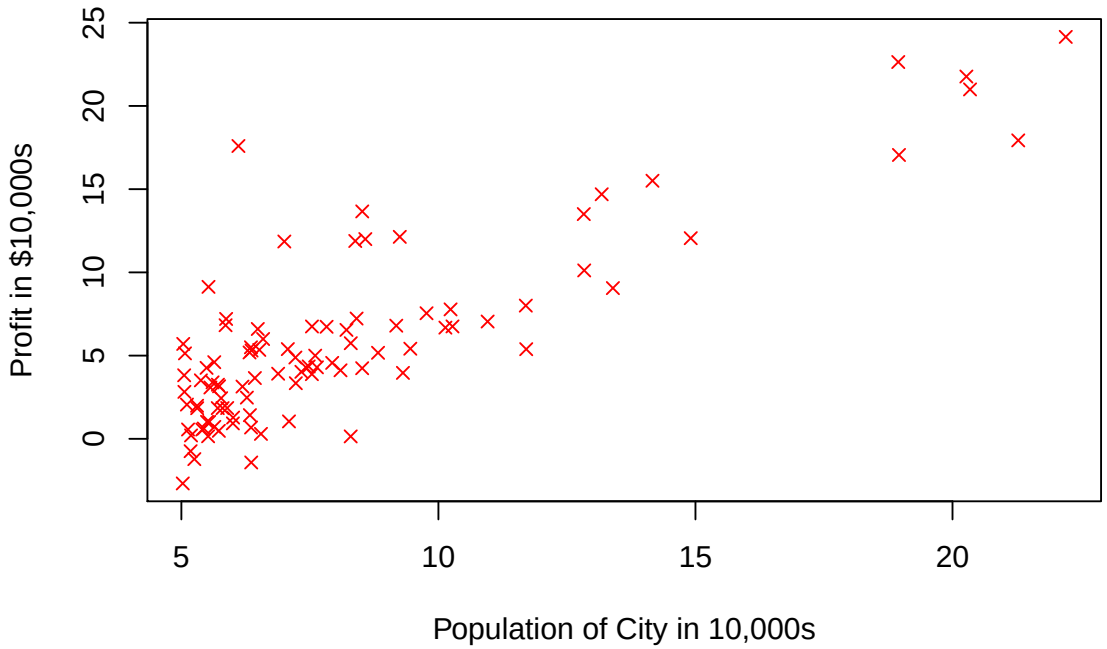
Next, the script calls the plotData function to create a scatter plot of the data. Your job is to complete plotData.R to draw the plot; modify the file and fill in the following code:

```
  plot(X, y, type="p",  pch=4, col="red",
       xlab = 'Population of City in 10,000s',
       ylab = 'Profit in $10,000s', cex=0.9);
```

Now, when you continue to run ex1.m, our end result should look like Figure 1, with the same red "x" markers and axis labels.

To learn more about the plot command, you can type help plot at the R command prompt or to search online for plotting documentation. (To change the markers to red "x", we used the option col="red' together with the plot command.

```
## [1] "Plotting Data ..."
```



[Figure 1: Scatter plot of training data]

3.2 Gradient Descent

In this part, you will fit the linear regression parameters θ to our dataset using gradient descent.

3.2.1 Update Equations

The objective of linear regression is to minimize the cost function

$$J(\theta) = \frac{1}{2m}\sum_{1=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

where the hypotesis $h_\theta(x)$ is given by the linear model

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

Recall that the parameters of your model are the θ_j values. These are the values you will adjust to minimize cost $J(\theta)$. One way to do this is to use the batch gradient descent algorithm. In batch gradient descent, each iteration performs the update

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{1=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} (\ simultaneously\ update\ \theta_j\ for\ all\ j).$$

With each step of gradient descent, your parameters θ_j come closer to the optimal values that will achieve the lowest cost $J(\theta)$.

> **Implementation Note**: We store each example as a row in the the X matrix in R. To take into account the intercept term (θ_0), we add an additional first column to X and set it to all ones. This allows us to treat (θ_0) as simply another 'feature'. Something

3.2.2 Implementation

In *ex1.R*, we have already set up the data for linear regression. In the following lines, we add another dimension to our data to accommodate the θ_0 intercept term. We also initialize the initial parameters to 0 and the learning rate alpha to 0.01.

```
X = cbind(rep(1,m) , data[,1]); ##### Add a column of ones to x
theta = rep(0,2); # initialize fitting parameters

# Some gradient descent settings
iterations <-1500;
alpha <-  0.01;
```

3.2.3 Computing the cost $J(\theta)$

As you perform gradient descent to learn minimize the cost function $J(\theta)$, it is helpful to monitor the convergence by computing the cost. In this section, you will implement a function to calculate $J(\theta)$ so you can check the convergence of your gradient descent implementation.

Your next task is to complete the code in the file *computeCost.R,* which is a function that computes $J(\theta)$. As you are doing this, remember that the variables X and y are not scalar values, but matrices whose rows represent the examples from the training set.

Once you have completed the function, the next step in *ex1.R* will run computeCost once using θ initialized to zeros, and you will see the cost printed to the screen.

You should expect to see a cost of 32.07.

3.3 Normal Equations

In the lecture videos, you learned that the closed-form solution to linear regression is

$$\theta = (X^T X)^{-1} X^T y$$

Using this formula does not require any feature scaling, and you will get an exact solution in one calculation: there is no "loop until convergence" like in gradient descent.

Complete the code in **normalEqn.R** to use the formula above to calculate θ. Remember that while you don't need to scale your features, we still need to add a column of 1's to the X matrix to have an intercept term (θ_0).

The code in **ex1.R** will add the column of 1's to X for you.

You should now submit the normal equations function

Optional (ungraded) exercise: Now, once you have found θ using this method, use it to make a price prediction for a 1650-square-foot house with 3 bedrooms. You should nd that gives the same predicted price as the value you obtained using the model t with gradient descent (in Section 3.2.1)

4 Submission and Grading

After completing various parts of the assignment, be sure to use the submit function system to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.

Part	**Submitted File**	**Points**
Warm up exercise	**warmUpExercise.R**	10 points
Compute cost for one variable	**computeCost.R**	40 points
Gradient descent for one variable	**gradientDescent.R**	50 points
Total Points		100 points

Extra Credit Exercises (optional)

Feature normalization	**featureNormalize.R**	10 points
Compute cost for multiple variables	**computeCostMulti.R**	15 points
Gradient descent for multiple variables	**gradientDescentMulti.R**	15 points
Normal Equations	**normalEqn.R**	10 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration. To prevent rapid-fire guessing, the system enforces a minimum of 5 minutes between submissions.