Implementation (25 points)

In this exercise, you are asked to run an experimental evaluation of linear regression and gradient descent algorithm on the Athens houses dataset. The input data is available in the *data* folder. Write your code in Python files **simple.py** and **multiple.py** in the *code* folder.

1. [Feature Scaling, 5 points]

To improve the convergence of gradient descent, it is important that the features are scaled so that they have similar ranges. Implement the standard scaling method using the skeleton code provided in the Python files.

2. [Simple Regression, 10 points]

Train a simple linear regression model to predict house prices as a function of their floor size, by running gradient descent for 200 epochs with a learning rate of 0.1. Use the dataset from the folder *linear_regression/data/simple*. Training data is in **train.txt** and test data is in **test.txt.** After training print the parameters and report the RMSE and the objective function values on the training and test data. Plot the training examples using the default blue circles and test examples using lime green triangles. On the same graph also plot the linear approximation. Plot J(w) vs. the number of epochs in increments of 10 (i.e. after 0 epochs, 10 epochs, 20 epochs, ...). Write your code in *linear_regression/code/simple.py.*

3. [Multiple Regression, 10 points]

Train a multiple linear regression model to predict house prices as a function of their floor size, number of bedrooms, and year. Run gradient descent for 200 epochs with a learning rate of 0.1. Use the dataset from the folder *linear_regression/data/multiple*. After training print the parameters and report the RMSE and the objective function values on the training and test data. Plot J(w) vs. the number of epochs in increments of 10 (i.e. after 0 epochs, 10 epochs, 20 epochs, ...). Compare the test RMSE with the one from the simple case above. Write your code in *linear_regression/code/multiple.py.*

Implementation details:

1. Feature Scaling.

Standardize the features of the examples by subtracting their mean and dividing by their standard deviation as follows: S = (X – mean)/std.
X contains the features of the examples, where each row is an example.
mean contains the means of the features, where each column is the mean of a feature
std contain the standard deviation of the features, where each column is the standard deviation of a feature.

2. Simple Regression.

a. Each column in the data provided corresponds to a data feature except the last column, which corresponds to the label/output which is house price. Each row in the data provided corresponds to an example.
Read in the data features to matrix X and data labels to t from the text file.

b. Implement gradient descent in the 'train' function to compute w = [w0 w1]. Use the 'compute_gradient' method to compute the derivative of the cost function, grad. Then update w = w – eta.grad in each epoch. Run gradient descent for 200 epochs, computing and storing the cost function every 10 epochs.

c. Use the 'compute_cost' function to compute and return the cost function. The cost function is:

$$(1/2N).\sum_{i=1}^{N}(h_w(x^{(i)}) - y^{(i)})^2$$

Where, N is the number of training examples.
$h_w(x^{(i)})$ is the predicted output of linear regression for example i, given by $w_0 + w_1.x^{(i)}$ in X
$y^{(i)}$ is the actual output of the example i, given in labels t

d. Use the 'compute_gradient' method to compute the derivative of the cost function, grad =

$$(1/N).\sum_{i=1}^{N}(h_w(x^{(i)}) - y^{(i)}).x_j^{(i)}$$

Where, grad = [$\partial/\partial w_0.J(w)$ $\partial/\partial w_1.J(w)$]
N is the number of training examples.
$h_w(x^{(i)})$ is the predicted output of linear regression for example i, given by $w_0 + w_1.x^{(i)}$ in X.
$x_j^{(i)}$ is feature j of example i in X, where j = 0,1
$y^{(i)}$ is the actual output of the example i, given in labels t

e. Use the 'compute_rmse' function to compute and return the RMSE on the dataset as square root of the cost function.

f. In the main method, standardize the training and test data, Xtrain and Xtest. Then add a column of ones for the bias features to the training and test data, as the first column. For example, given:

$$\text{Xtrain} = \begin{bmatrix} 3032 \\ 2074 \\ . \\ . \end{bmatrix}, \text{ after adding the bias features, Xtrain} = \begin{bmatrix} 1 & 3032 \\ 1 & 2074 \\ 1 & . \\ 1 & . \end{bmatrix}$$

3. Multiple Regression.

Implement the above steps for the multiple linear regression problem where there are 3 data features corresponding to the first 3 columns and the labels/output corresponding to the last column in the dataset. Compare the test RMSE with the one from the simple case above.

4. BONUS. [Stochastic gradient descent (SGD), 10 points]

Implement SGD and run it for problems 1, 2 and 3 above using the same learning rate, eta, and number of epochs. Use the 'train_SGD' function to implement SGD and obtain the same results as you did for batch GD (parameters, RMSE, cost, linear approximation and plot of cost vs. epochs) using the provided code. Compare the SGD solution to the batch GD solution. Does SGD need fewer or more epochs to arrive at the same parameters as batch GD. If fewer, how many epochs is sufficient?

Submission:

Only modify the Python files to implement your code and submit your code with the assignment folder along with a Report file containing the results and discussion of running your code. The Report should contain results of running your code, including terminal outputs and plots, as well as discussion/explanation of results asked above. The Report should be in docx or pdf format in the main directory of the assignment folder. You may include the plots generated from running the code in your submission.

Please observe the following when submitting your assignment:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. Make sure your code runs correctly when used in the directory structure of the assignment folder.
4. I will compare your results with the results I got from running the code to determine accuracy.