Gradient Descent Implementation

Doran Wu

October 17, 2021

A. ABSTRACT

This report details an implementation of gradient descent for linear regression. Linear regression could work for both multivariate and univariate cases and also the ordinary least squares case or closed form solution of the problem.

B. DESCRIPTION OF PROBLEM

From the dataset taken from link. the goal is to predict Concrete Compressive Strength from 8 potential independent variables: Cement Component, Blast Furnace Slag, Fly Ash, Water Component, Superplasticizer, Coarse Aggregate, Fine Aggregate, Age. Methods to predict Concrete Compressive Strength in this report include univariate linear regression, multivariate linear regression, and ordinary least squares regression.

C. RUN IT YOURSELF

To test the program yourself follow these steps...

1. Prior to running package, ensure system has at least Python 3.7 downloaded and these packages...
   - pandas
   - numpy
   - matplotlib
2. To run the program open the command-line interpreter for your respective machine (terminal for mac or cmd.exe for windows)
3. Then navigate to the location which holds the folder of files that were downloaded from the aforementioned link
4. program can be invoked with 'python main.py', however the program requires 11 additional command line arguments which include in order...
   - excel file name in quotes (string in quotes)
   - number of training instances (int)
   - linear (string)
   - what independent variables to use in quotes(list of comma separated ints)
   - what is the dependent variable or response (int)
   - learning rate in quotes (scientific notation float)
   - number of epochs or generations (int)
   - 'penalty' or 'no' (string)
   - 'clip' or 'no' (string)
   - 'graph' or 'no' (string)
   - 'normalize' or 'no' (string)
5. One example to input in command-line interpreter is 'python main.py 'Concrete_Data.xls' 900 linear '1,6,7' 9 '1.0E-8' 500 penalty no no no'
6. The example will run multivariate linear regression with 3 independent variables (cement component, coarse aggregate, fine aggregate), dependent variable (concrete compressive strength), with a learning rate of $1.0E08$, 500 generations, penalty, no gradient clipping, no graphing, and no normalizing

D. NOTES

- To input the desired predictors or response, features go from 1-9 inclusive. For example if you wanted to select the concrete compressive strength you would input '9', age would be '8', and so on
- All initial thetas are set to 0
- Number of training instances is kept constant at 900 points
- Number of test instances is kept constant at 130 points
- Gradient Clipping is not used for the analysis
- Plots of trained uni-variate models on top of scatterplots refer to the data table above them and are taken at an epoch of 1000
- If user selects the graph option for univariate linear regression, two graphs will be given: a plot showing the current mean squared error after each epoch, and a plot showing the trained uni-variate model after a given epoch on top of a scatterplot for the training data
- Models run to 1000 epochs, however the recorded MSE at 1000 epochs for both the train and test data is not necessarily the minimum for the predictor/s
- For the gradient descent algorithms, learning rate is kept constant throughout training and thetas are updated after getting the sum from all training instances and then updating thetas once every epoch or generation

E. DATA PROCESSING

The pandas and numpy packages are utilized to process data in the given excel sheet. Potential predictor variables are put into a dictionary (functional for both univariate and multivariate analysis) and the single response variable's data is put within a one-dimensional array.

In order to reduce the margin of potential learning rates and improve the accuracy of models, one method that the user can decide is whether they want to normalize their predictor variable data or not. Within the data sections it is apparent normalization improves the accuracy of models by a significant margin. One potential method of normalizing data with the mean and standard deviation is implemented with this equation...

$$X_{normalized} = \frac{x' - x_{mean}}{x_{std}}$$

Each data point in the predictor will be normalized by taking subtracting the mean of the predictor variable data from the original data point and then dividing it by the standard deviation for the predictor variable.

F. METRICS

Mean Squared Error

Mean Squared Error (MSE) is used to assess the efficacy of the models and is also the loss function for gradient descent. MSE is implemented with the general pseudocode below...

```
Output: $mse$
1: $total \leftarrow 0$                                   ▷ initialize value
2: for $i$ in $response$ do                               ▷ iterate through y-values
3:     $error \leftarrow (i - (\theta_0 + ... + \theta_n x))^2$    ▷ difference between actual and prediction
4:     $total = total + error$                            ▷ update total
5: $mse = total/(response\ range)$                        ▷ divide total error by number of instance
```

G. CHOOSING A LEARNING RATE - UNIVARIATE AND MULTIVARIATE

For the models, I employed a similar technique to the common learning rate annealing strategy of gradually lowering a high learning rate as model is being trained. The two criteria that were used to choose a learning rate is whether the MSE explodes which suggests the learning rate chosen is too high or the MSE barely improves even at the start which means the learning rate is too small.

Therefore, after some experimenting with learning rates, these are the general learning rates I have chosen...

- $1.0E^{-8}$ for non-normalized univariate linear regression
- $1.0E^{-2}$ for normalized univariate linear regression
- $1.0E^{-8}$ for non-normalized multivariate linear regression
- $1.0E^{-3}$ for normalized multivariate linear regression

LINEAR REGRESSION - IMPLEMENTATION

H. General Mathematic Methodology - Univariate

H.1 The standard linear regression formula is:

$$y_i = \theta_0 + \theta_1 x$$

The y_i = output, θ_0 = intercept or first weight, θ_1 = second weight, x = input.

H.2 Using Mean Squared Error (MSE) as the cost function:

$$C(\theta_0, \theta_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - pred_i)^2$$

$C(\theta_0, \theta_1)$ = sum of the mean squared error (difference between actual and predicted outputs) squared for thetas 0 and 1, n = number of data points, y_i = is the actual output or value in the test set, $pred_i$ = is the predicted value calculated from the standard regression formula.

H.3 Updating thetas with gradient descent:

In order to update thetas—θ_0, θ_1, we apply the general gradient descent formula to update both thetas.For some theta of c (either θ_0, θ_1 for this case) we can update the theta by subtracting the learning rate multiplied by the gradient of the cost function or partial derivative for the formula in H.2.

$$\theta_c = \theta_c - \alpha \frac{\partial}{\partial \theta_c} C(\theta_0, \theta_1)$$

H.4 Partial derivative of the cost function for θ_0

$$\frac{\partial}{\partial \theta_0}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - (\theta_0 + \theta_1 x))^2\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial \theta_0}((y_i - (\theta_0 + \theta_1 x))^2)$$

$$= \frac{1}{n}\sum_{i=1}^{n}2(y_i - (\theta_0 + \theta_1 x))\frac{\partial}{\partial \theta_0}(y_i - (\theta_0 + \theta_1 x))$$

$$= \frac{1}{n}\sum_{i=1}^{n}-2(y_i - (\theta_0 + \theta_1 x))$$

H.5 Partial derivative of the cost function for θ_1

$$\frac{\partial}{\partial \theta_1}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - (\theta_0 + \theta_1 x))^2\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial \theta_1}((y_i - (\theta_0 + \theta_1 x))^2)$$

$$= \frac{1}{n}\sum_{i=1}^{n}2(y_i - (\theta_0 + \theta_1 x))\frac{\partial}{\partial \theta_1}(y_i - (\theta_0 + \theta_1 x))$$

$$= \frac{1}{n}\sum_{i=1}^{n}-2x(y_i - (\theta_0 + \theta_1 x))$$

H.6 Combine equations from H.3, H.4, H.5 into compact form

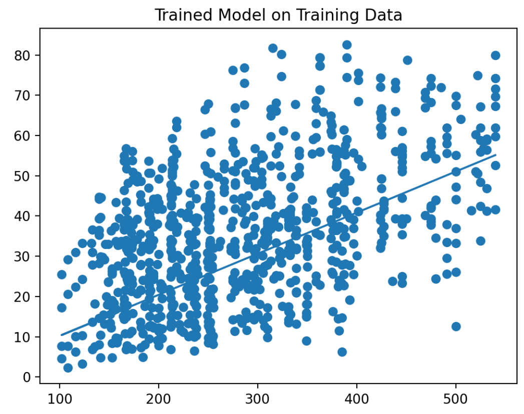
$$\theta_0 = \theta_0 - \alpha\left(\frac{1}{n}\sum_{i=1}^{n}-2(y_i - (\theta_0 + \theta_1 x))\right)$$

$$\theta_1 = \theta_1 - \alpha\left(\frac{1}{n}\sum_{i=1}^{n}-2x(y_i - (\theta_0 + \theta_1 x))\right)$$

Now we can write the pseudocode for both univariate and multivariate linear regression.

I. Univariate Pseudocode - Linear Regression

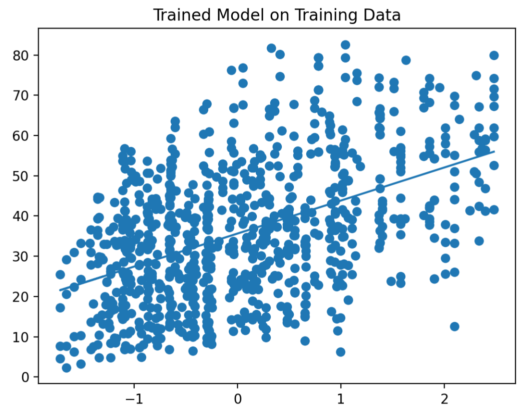
```
Output: $\theta_0, \theta_1$
1: for $epoch$ in $epochs$ do                                  ▷ Stopping: Finish generations
2:     $grad_{\theta_0} \leftarrow 0, grad_{\theta_1} \leftarrow 0,$                    ▷ reset gradients
3:     for $instance$ in $instances$ do                         ▷ Sum aspect for gradient
4:         $cost \leftarrow actual - (\theta_0 + \theta_1 * x)$            ▷ linear regression formula
5:         $grad_{\theta_0} \leftarrow grad_{\theta_0} + (-2 * cost)$        ▷ updating gradient
6:         $grad_{\theta_1} \leftarrow grad_{\theta_1} + ((-2 * x) * cost)$    ▷ updating gradient
7:     $\theta_0 \leftarrow \theta_0 - (\alpha * grad_{\theta_0})$                ▷ updating intercept
8:     $\theta_1 \leftarrow \theta_1 - (\alpha * grad_{\theta_1})$                ▷ updating theta
```



J. Data - Univariate Linear Regression

Cement Component

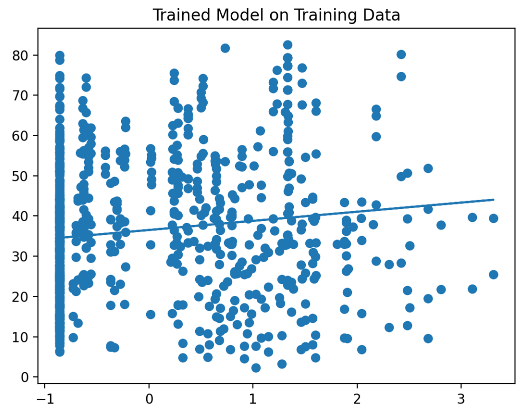
Data - with normalization		
Epochs	Train MSE	Test MSE
200	228.7077	83.2191
500	228.3311	81.1422
1000	228.3311	81.1388



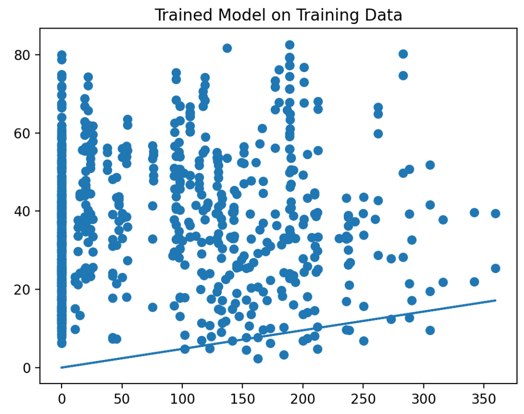
Data - no normalization		
Epochs	Train MSE	Test MSE
200	887.5606	667.0519
500	452.3626	338.0371
1000	279.45685	171.8694

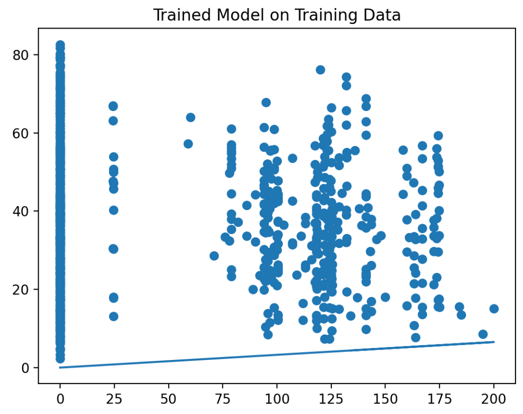
Blast Furnace Slag

Data - with normalization		
Epochs	Train MSE	Test MSE
200	290.7733	159.6951
500	290.7733	159.6738
1000	290.7732	159.6951



Data		
Epochs	Train MSE	Test MSE
200	1563.9806	1085.1775
500	1490.8066	979.0028
1000	1390.0785	833.9284



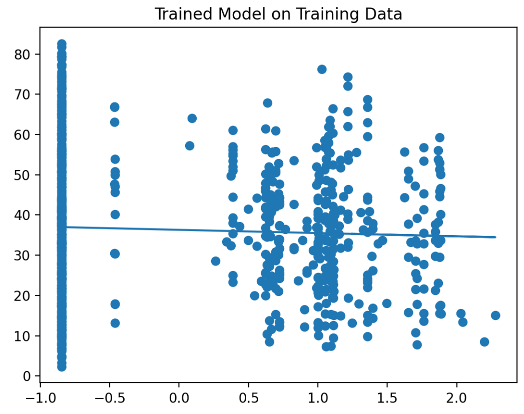


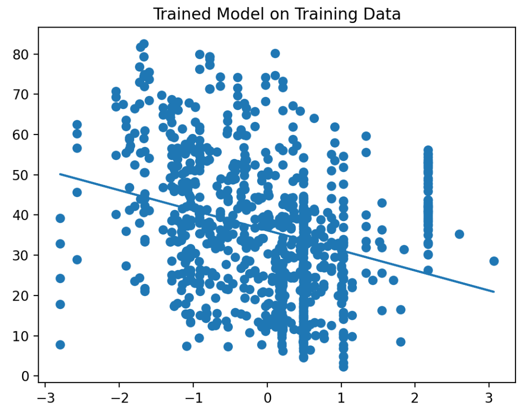
Fly Ash

Water

Data - with normalization		
Epochs	Train MSE	Test MSE
200	295.7353	141.5220
500	295.2626	149.4225
1000	295.2626	149.4662

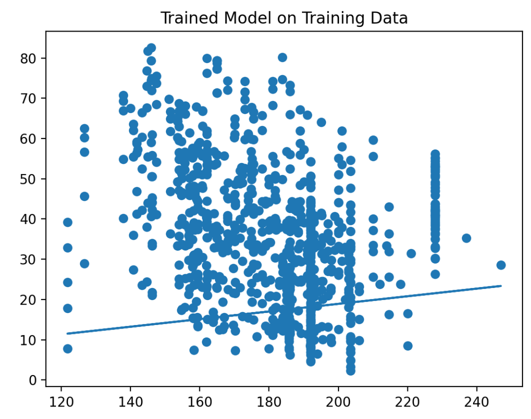
Data - with normalization		
Epochs	Train MSE	Test MSE
200	270.5002	151.8437
500	270.1026	154.9825
1000	270.1026	154.9904

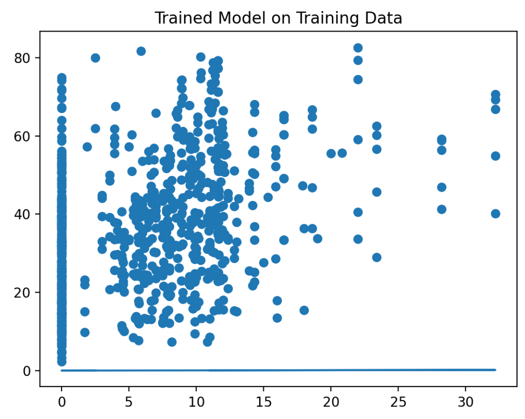




Data - no normalization		
Epochs	Train MSE	Test MSE
200	1595.4673	1132.0681
500	1562.3196	1085.809
1000	1512.2947	1017.5675

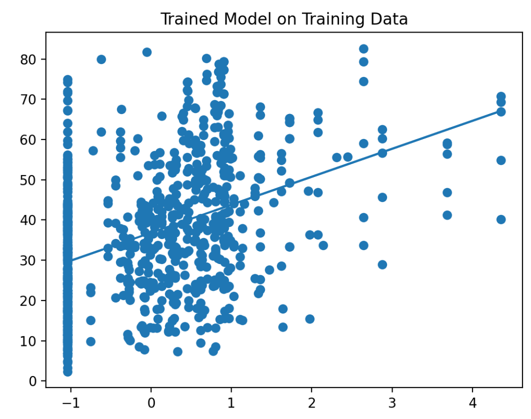
Data - no normalization		
Epochs	Train MSE	Test MSE
200	1325.3215	897.2670
500	1007.9429	616.3292
1000	693.0428	354.5050





SuperPlasticizer

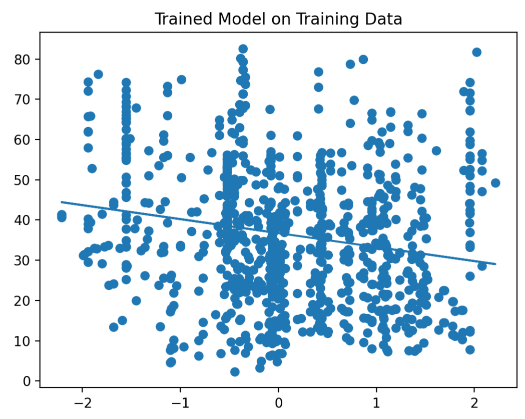
Data - with normalization		
Epochs	Train MSE	Test MSE
200	244.8319	221.4914
500	244.3012	235.2943
1000	244.3012	235.3382



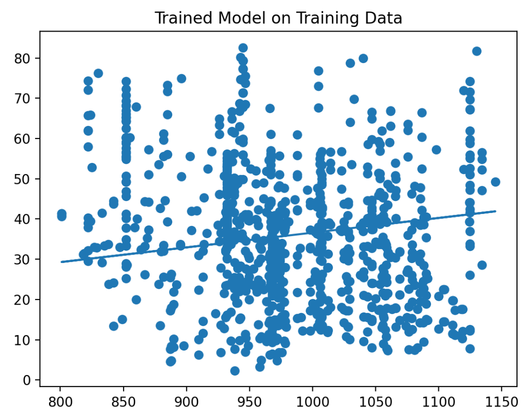
Data - no normalization		
Epochs	Train MSE	Test MSE
200	1618.4503	1164.7480
500	1617.6512	1163.8763
1000	1616.3209	1162.4252

Coarse Aggregate

Data - with normalization		
Epochs	Train MSE	Test MSE
200	285.1179	176.1472
500	284.4839	190.1937
1000	284.4839	190.2678

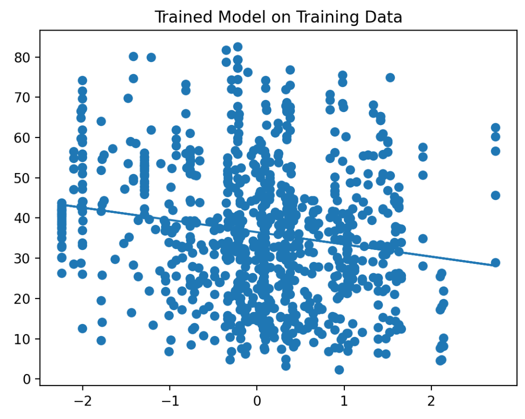


Data - no normalization		
Epochs	Train MSE	Test MSE
200	322.8659	163.7674
500	322.3460	165.9489
1000	322.3459	165.9564

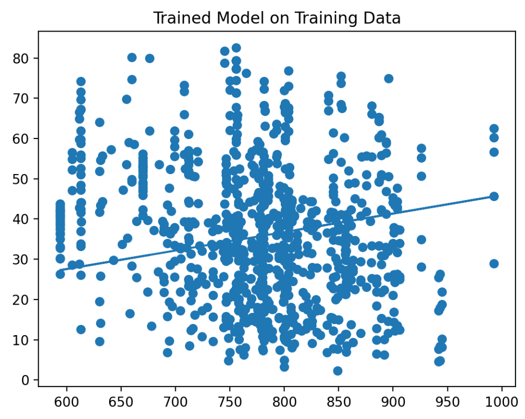


Fine Aggregate

Data - with normalization		
Epochs	Train MSE	Test MSE
200	286.7517	162.1475
500	286.3124	168.9817
1000	286.3123	169.0007

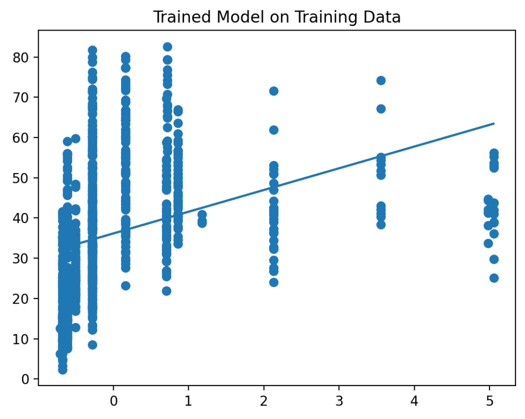


Data - no normalization		
Epochs	Train MSE	Test MSE
200	342.9796	161.6019
500	333.4429	171.0541
1000	333.4368	171.5260

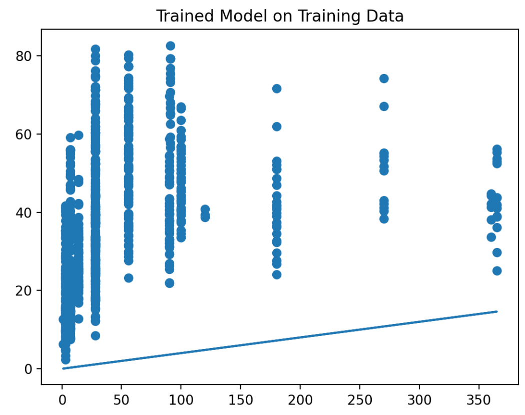


Age

Data - with normalization		
Epochs	Train MSE	Test MSE
200	263.3083	147.9067
500	262.9107	150.9166
1000	262.9107	150.9252



Data - no normalization		
Epochs	Train MSE	Test MSE
200	1583.3410	1150.2662
500	1533.4019	1128.6279
1000	1458.7556	1094.9570



K. Analysis from Data - Univariate Linear Regression

Given the data accumulated from Section J, there are a few conclusions that can be drawn. Even if the model's mean squared error for the training data is improving between epochs, that does not necessarily mean the mse for the test data is improving. One example is when the predictor is age and the data is normalized. While, the train mse is improving (lowering), the test mse is increasing.

Generally speaking, disregarding the different learning rates, models with normalized data perform better when the data is normalized.

There will be two primary criteria for choosing whether a feature is considered important or not important: whether the trained model lines for both normalized and non-normalized data look like they fit the data, and whether the test and train mse for a given feature (for both train and test) is small compared to the mse for other features. Given these criteria, it appears the most important feature is Cement Component because of the lowest mse for both the train and test data and how the trained model fit the training data well. Other important features could include Coarse Aggregate and Fine Aggregate. While their respective mse is not lower compared to others for when data is normalized, their model still appears to fit the test data well for when data is not normalized and when it is normalized.

L. General Mathematic Methodology - Multivariate

L.1 The standard linear regression formula is:

$$y_i = \theta_0 + \theta_1 x + \theta_2 x + \theta_3 x + \theta_4 x + \theta_5 x + \theta_6 x + \theta_7 x + \theta_8 x$$

For more than one independent variable the linear regression formula is similar to the one found in H.1 but with more thetas.

L.2 Updating thetas with gradient descent:

In order to update multiple thetas we apply the same general steps found in H.4 with univariate linear regression. But this time instead of only taking the partial derivatives for two thetas (intercept and slope) we will have more slopes. The partial derivatives for each subsequent theta will be similar to H.6 because when you take partial derivatives for a specific variable, it becomes 1 while the others become 0. It could be visualized with this formula.

For $\theta_j : j \in \{1, \infty\}$, the updated θ_j can be:

$$\theta_j = \theta_j - \alpha(\frac{1}{n}\sum_{i=1}^{n} -2x(y_i - (\theta_0 + \theta_1 x ... \theta_j x)))$$

M. Multivariate Pseudocode - Linear Regression

```
Output: $\theta_0 ... \theta_n$
1:  for $epoch$ in $epochs$ do
2:      $grad_{\theta_0} ... grad_{\theta_n} \leftarrow 0$
3:      for $instance$ in $instances$ do
4:          $cost \leftarrow actual - (\theta_0 + \theta_1 x + ... + \theta_n x)$
5:          for $gradient$ in $gradients$ do
6:              if $gradient \leftarrow 0$ then
7:                  $gradient \leftarrow gradient + (-2 * error)$
8:              else
9:                  $gradient \leftarrow gradient + (-2 * x * error)$
10:     for $gradient$ in $gradients$ do
11:         $gradient \leftarrow gradient / (num\ instances)$
12:     for $theta$ in $range(theta)$ do
13:         $thetas[theta] \leftarrow theta - alpha * gradients[theta]$
```

N. Data - Multivariate Linear Regression

From the univariate linear regression we were able to see the three most important features (those with lowest test MSE for 1000 epochs at an $\alpha = 1.0E^-2$ or $\alpha = 1.0E^-8$ was Cement Component, Coarse Aggregate, and Fine Aggregate). Therefore we will run the multivariate linear regression on all the independent variables, multivariate linear regression on only the important features to check which one will give us the lowest MSE on both the training and test data, and finally multivariate linear regression on all the unimportant features.

Independent Variables - Cement Component, Coarse Aggregate, Fine Aggregate

Data - with normalization		
Epochs	Train MSE	Test MSE
200	865.2816	575.2620
500	570.9470	271.9622
1000	1526.4844	956.3885

Data - no normalization		
Epochs	Train MSE	Test MSE
200	547.0926	420.7397
500	351.2986	215.5658
1000	309.1798	146.8988

Independent Variables - Cement Component, Blast Furnace Slag, Fly Ash, SuperPlasticizer, Water, Coarse Aggregate, Fine Aggregate, Age

Data - no normalization		
Epochs	Train MSE	Test MSE
200	788.6435	600.2844
500	655.8773	290.3609
1000	12507.9587	6099.1979

Data - with normalization		
Epochs	Train MSE	Test MSE
200	297.6278	147.3659
500	285.6147	138.3263
1000	268.5266	125.3133

O. Analysis from Data - Multivariate Linear Regression

Normalization was able to improve the mse the univariate linear regression models, however it appears the antithesis occured for multivariate linear regression. This could probably be attributed to a chosen learning rate that is too high for these cases. Take for example data when all potential predictors are used for regression. After 500 epochs the train mse and test mse are 655.8773 and 290.3609 respectively. One would expect that at 1000 epochs the train and test mse would be lower, however they jump substantially to 12507.958 and 6099.1979 for the train and test mse. Because the learning rate is too high, the model makes large jumps rather than small steps which could explain this phenomenon.

It appears multivariate regression with all eight variables performed better compared to multivariate with only three important features because it achieved the lowest train and test mse pair. However this could change if more epochs are used to train the model and a lower learning rate is utilized.

The best univariate model with cement component and normalized data performed better than the two multivariate models (however could change with different learning rate and more epochs).

P. GENERAL MATHEMATIC METHODOLOGY - LEAST SQUARES

$$\theta_1 = \frac{n * \sum_{i=1}^{n} xy - \sum_{i=1}^{n} x \sum_{i=1}^{n} y}{n * \sum_{i=1}^{n}(x^2) - (\sum_{i=1}^{n} x)^2}$$

$$\theta_0 = \frac{\sum_{i=1}^{n} y - m * \sum_{i=1}^{n} x}{n}$$

Utilizing these formulas we can get the intercept and slope (or thetas) for a given independent variable and dependent variable. It can be implemented with this pseudocode...

```
Output: $mse_{train}, mse_{test}$
1: $x_{squaresum} \leftarrow sum(x^2\ for\ all\ x)$  ▷ calculate the sum of all $x^2$
2: $x_{sum} \leftarrow sum(x\ for\ all\ x)$  ▷ calculate the sum of all $x$
3: $y_{sum} \leftarrow sum(y\ for\ all\ y)$  ▷ calculate the sum of all $y$
4: $xy_{sum} \leftarrow sum(x * y\ for\ all\ x\ and\ y)$  ▷ calculate the sum of all $xy$
5: $\theta_1 \leftarrow (training_range * xy_{sum} - x_{sum} * y_{sum})/(train\_range * x_{squaresum} - (x_{sum})^2)$  ▷ calculate slope
6: $\theta_0 \leftarrow (y_{sum} - \theta_1 * x_{sum})/(train\_range)$  ▷ calculate intercept
7: $mse_{train} = calcMSE(thetas, train\_data)$  ▷ calculate mse for training data, use previously defined function
8: $mse_{test} = calcMSE(thetas, test\_data)$  ▷ calculate mse for test data
```

Q. Data - Ordinary Least Squares Regression

Independent Variables - Cement Component

Data - no normalization	
Train MSE	Test MSE
228.3311	81.1388

Independent Variables - Blast Furnace Slag

Data	
Train MSE	Test MSE
290.7733	159.6951

Independent Variables - Fly Ash

Data	
Train MSE	Test MSE
295.2626	149.4662

Independent Variables - Water

Data	
Train MSE	Test MSE
270.1026	154.9904

Independent Variables - SuperPlasticizer

Data	
Train MSE	Test MSE
244.3012	235.3382

Independent Variables - Coarse Aggregate

Data	
Train MSE	Test MSE
284.4839	190.2678

Independent Variables - Fine Aggregate

Data	
Train MSE	Test MSE
286.3124	169.0007

Independent Variables - Age

Data	
Train MSE	Test MSE
262.9107	150.9252

R. Analysis from Data - Ordinary Least Squares Regression

Just to note, normalizing data will not affect the output for least squares regression. The train and test mse will stay constant if data is normalized or not.

The performance with the OLS method was slightly different from the results attained in section J. However, this is probably because of the hyperparameters and number of generations utilized for the univariate models. It appears that in many cases, the normalized data was able to get close to the values attained with the closed form method. For example, the test mse for the univariate linear regression rounded to the same number as the value attained when using OLS.

S. Regularization - Extra Credit

The regularization technique that will be implemented is Lasso Regularization or the L1 norm. It is implemented with this general formula.

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - pred_i)^2 + \lambda\sum_{i=1}^{n}\theta_j$$

This will help shrink the weights to 0 in order to mitigate the probability of overfitting. To implement this in the code, we apply apply the penalty to the gradient term, here is a pseudocode which shows this in action:

```
1: $penalty \leftarrow \frac{\lambda}{range}$
2: for $theta$ in $thetas$ do                              ▷ iterate through thetas
3:     if $theta = 0$ then
4:         $theta \leftarrow theta - alpha * gradient$    ▷ if intercept don't apply penalty
5:     else
6:         $theta \leftarrow theta - alpha * (gradient + (penalty * theta))$    ▷ apply penalty to all non-intercept weights
```

The reason why you do not update the intercept is simply because it is not attributed to any predictor, while for any $\theta_{j>0}$, there is an x value attributed to it.

To show this implementation of regularization works, we will run multivariate linear regression with normalized data and two predictors: Cement Component and Blast Furnace Slag, a constant learning rate of $1.0E^{-4}$ in pairs and with max epochs going to 1000. One of the pair will have the penalty applied to it and the other will not have the penalty applied to it (apply penalty by switching the 9th command line argument to penalty if you want the penalty or to any other string for no penalty). If the implementation is correct, the regularized features should be closer to 0 versus the non regularized features. In addition, a low λ for regularization will only show a very small difference between regularized and non-regularized features therefore we will use a $\lambda = 1.0E^{6}$.

T. Data - Lasso Regularization

Independent Variables - Cement Component, Blast Furnace Slag

Data - normalized without regularization		
Train MSE	Test MSE	Thetas
1153.1641	822.1612	[6.5479, 1.5995, 0.0243]

Data - normalized with regularization		
Train MSE	Test MSE	Thetas
1185.9526	790.2033	[6.5374, 0.0079, 0.0036]

U. Analysis of Data - Lasso Regularization

As we can see with the data, when a penalty is applied, the non-intercept thetas are closer to 0. In addition, the regularized model performed better on the testing data than the non-regularized model. However, that is the opposite for the training data. This suggests that the learning rate, number of iterations, beginning thetas, etc. have an impact on whether or not the regularized model generally perform better than the non-regularized model. However, it is certain that with regularization the features are shrunk closer to 0.