Foundations of Data Science Assignment-2

Achyuta Krishna V	**2018A7PS0165H**
Jonathan Samuel J	**2018AAPS0460H**
Shubh Deep	**2018A7PS0162H**

Aim of the assignment :

To implement linear regression after adequate pre-processing of data using the three methods:

1. Solving by Linear Equations
2. Gradient Descent
3. Stochastic Gradient Descent

Pre-processing of Data

The pre-processing involved two steps:

- Standardization of Data
- Splitting of Data into Training and Testing Sets

Standardization

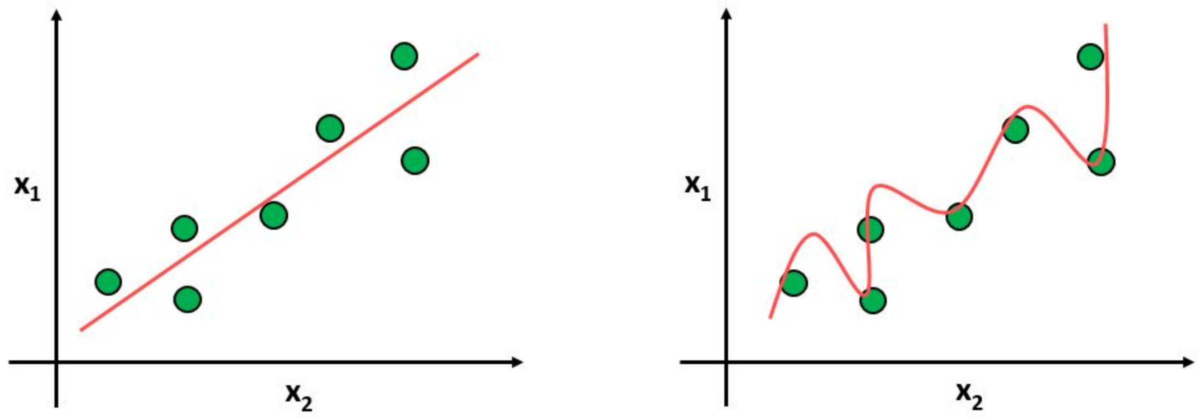
Standardization of data involves changing the values of the columns in the dataset to a common scale. This is done to eliminate the effect that the range of a feature may have on our result. For example, 2 features may have the range 0 - 100 and 20,000 - 500,000 respectively. These ranges are very different from each other and without any standardization, the value of feature 2 will influence the result much more than feature 1 due to its larger values. But this analysis is flawed as larger values don't necessarily mean more importance in the resulting model.

There are several methods to normalize a dataset. The method used here is **Min-Max Standardization.** This method rescales the range of the data to [0,1]. This helps us to compare different features on an equal footing.

$$\hat{X} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Splitting of Dataset

This step involves splitting of our data into two subsets: Training Dataset and Testing Dataset. We fit our model on the train data and make predictions on the test data. The model is developed on the feature-target values present in the training dataset. This model is then used to predict the outputs for feature values present in the training data. These prediction values are then compared with the corresponding target values. Error values are then calculated based on the difference between these values. This step helps us avoid overfitting our model to the data. Overfitting means that the model fits too closely to the dataset. Overfitting a model would result in a model that fits the training data too accurately and the model learns or describes the “noise” in the training data and ignores the actual relationships between the data. This problem is solved when the dataset is split into two sets. A model that overfits the data would result in an unusually high value of testing error.



The image on the left depicts an ideal linear regression model for the given data
The image on the right depicts an overfitting model for the given data

Implementation:

Normal Equations Method

The Normal Equations method is an analytical approach to Linear Regression which can arrive at an optimal value or the global minimum without going through multiple iterations of gradient descent. The normal equation is as follows:

$$\boxed{\theta = \left(X^T X\right)^{-1} . \left(X^T y\right)}$$

θ : hypothesis parameters
X : Input feature value of each instance.
Y : Output value of each instance.

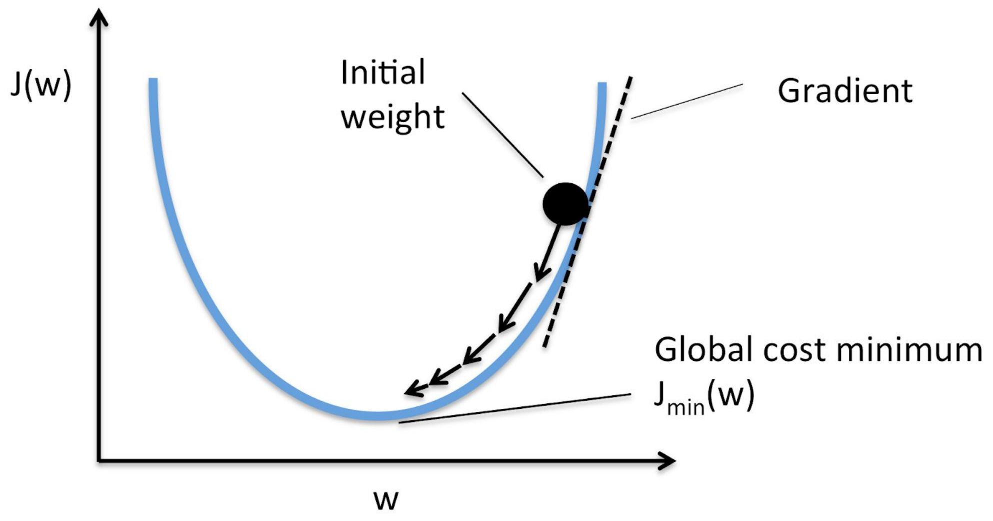
To implement this method, the dataset is read as a numpy matrix. A bias column of 1’s is added as the first column in the matrix. This matrix is then split into training and testing datasets which are further split into a feature matrix(X) and a target matrix(Y). After this the value of our hypothesis parameters are calculated using the above equation by passing the datasets to our normal model function. The transpose of the feature matrix, dot products of matrices and the inverse of the resulting matrix were calculated using inbuilt functions in the numpy library.

After training the model, the **θ** vector is used to calculate the outputs for feature values. These predicted values are then compared with the given target values to calculate the error and the Root Mean Square Error is calculated using vectorization and functions available in numpy library.

To obtain 20 different models we run a loop of 20 iterations. In each loop we randomly split the data, standardise it and then pass it to the above function to return the training and testing error for each model. We save these in an array and later use them to find the minimum, mean and variance of the testing and training errors.

Gradient Descent Method

Gradient Descent Method is an iterative approach to Linear Regression which can arrive at the optimal value or the global minimum by taking steps proportional to the negative of the gradient at the current point.



To implement this method we define a function to return us the feature derivative at each point.

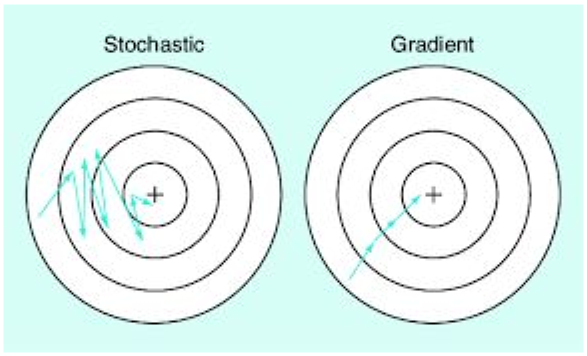
We then define another function which has two iterations. The outer iteration runs for the maximum iterations we pass in. So for each outer iteration we compute the predictions and in turn compute the errors. We then iterate through all the feature weights and calculate the derivative and then subtract the derivative*step size from the current weight. We also store the training error(for plotting) and output it every one tenth of the outer iteration. We then return the weights and the training error we saved before in this function.
(*Function name* :`regression_gradient_descent`)

In order to control the above function and to make use of the returned weights to use it with the testing set we have another function. So this function takes in the weights and computes the predicted values for testing set and stores the testing error as well as the training error. (*Function name*: `calcGradientDescent`)

Now we have a loop of 20 to obtain 20 different models. In each loop we randomly split the data, standardise it and then pass it to the above function to return the training and testing error for each model. We save these in an array and later use them to find the minimum, mean and variance of the testing and training errors.

Stochastic Gradient Descent Method

Stochastic Gradient Descent is also like gradient descent except here we select only one random data point in each iteration and use it to compute the gradient to reach the optimal value or the global minimum. This is done in order to decrease the computation power and time needed to train models with huge data sets.



So here we define a function which first shuffles the features along with target values so that it will help us to pick a random point later to find the derivative. We then have 2 iterations just like Gradient Descent method. Inside the outer iteration here we find the prediction value and the error of only the random data point instead of the data points like in Gradient Descent. We then iterate over all the weights and update them depending on the gradient we get from the error and the random data point. We then calculate and display the training errors just like before in GD. We also have shuffling inside the outer iteration just so that when we complete one full cycle of the points in the iterations we can shuffle and start again so that it will again be random. We then return the weights and training errors. (*Function name*: `stochastic_gradient_descent`)

The next function we have is to use the above function to compute the weights and then use them to calculate the training errors and testing errors and return them. (*Function name*: `calcStochasticGradientDescent`)

We then have a loop of 20 to create 20 different models. In each loop we randomly split the data, standardise it and then pass it to the above function to return the training and testing error for each model. We save these in an array and later use them to find the minimum, mean and variance of the testing and training errors.

Errors

The following table documents the minimum errors achieved by us using the three models

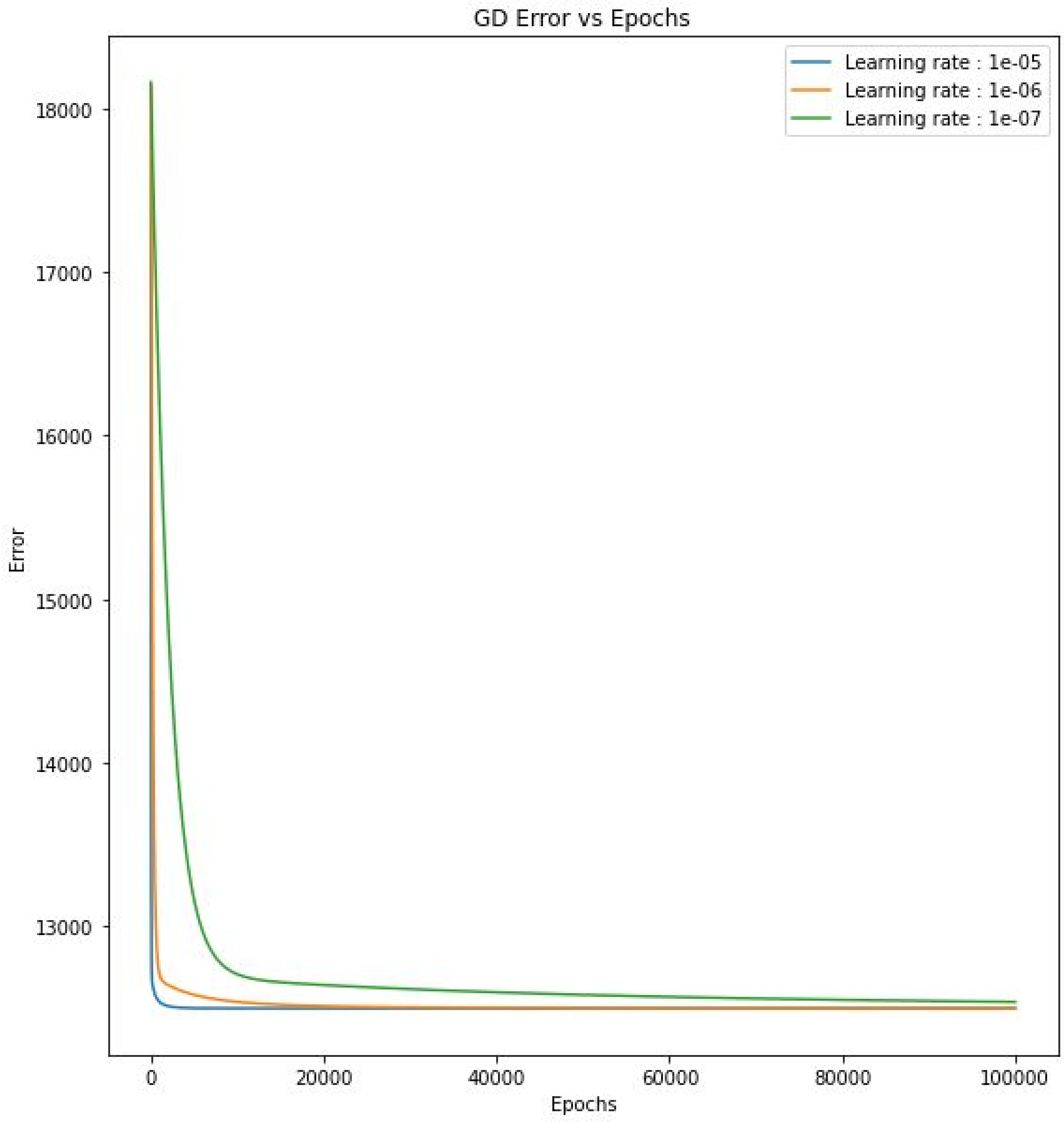
Normal Equations		Gradient Descent		Stochastic Gradient Descent	
Training	Testing	Training	Testing	Training	Testing
12066.6246 55721136	11752.5031 5210275	12178.5210 52716076	11731.1331 93993644	13264.5617 92512557	12686.6478 00639044

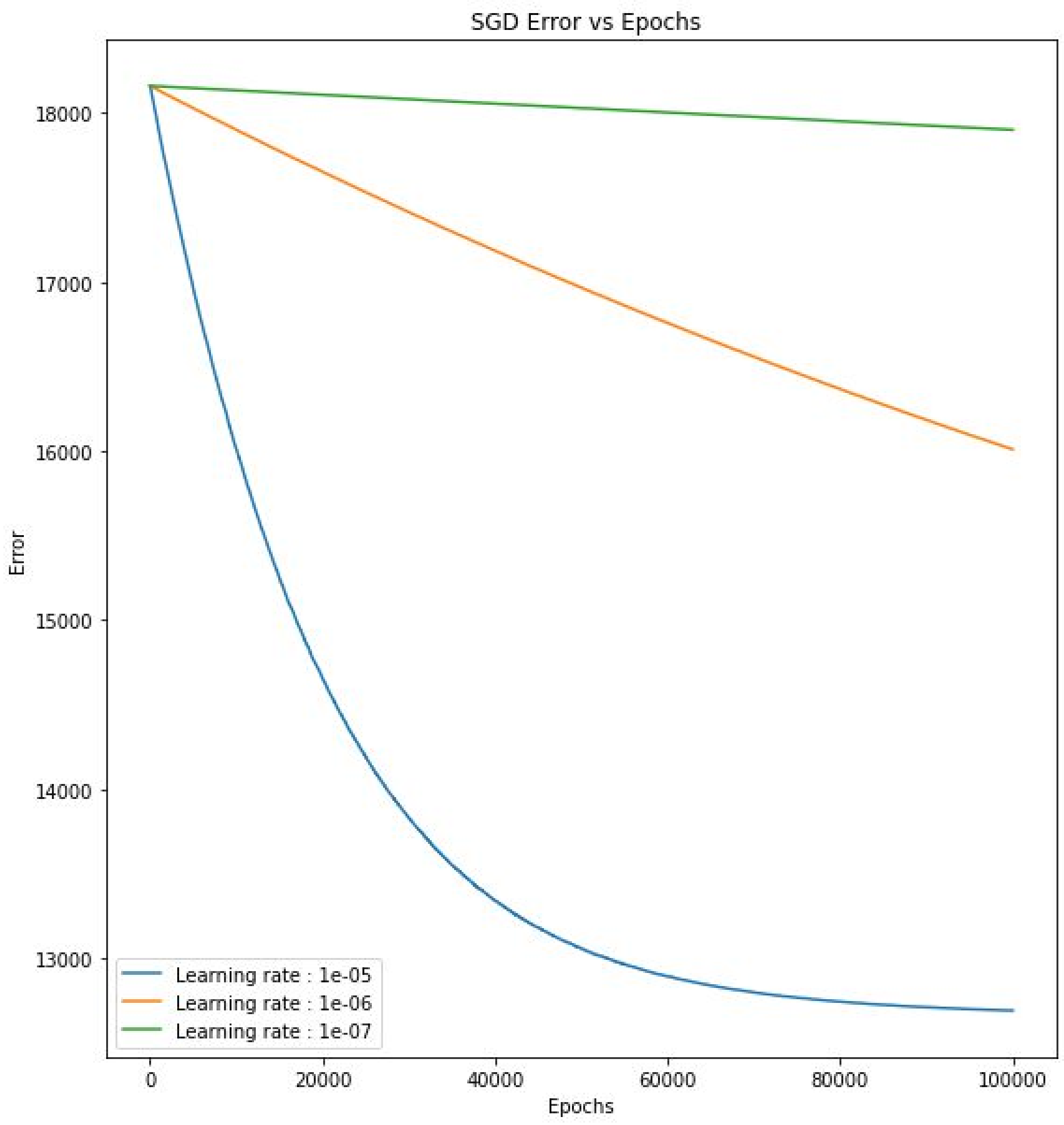
Observations

The Normal Equations Method gives the most optimal solution. This is because the normal equation method is a purely analytical process. On the other hand, the gradient descent methods are an iterative approach the accuracy of which depends on the learning rate chosen by us.

Comparing between the gradient descent methods, Gradient descent results in an more optimal solution than Stochastic Gradient Descent method.This can be attributed to the fact that the update done to the features in SGD depends completely on the batch size and the shuffling of the dataset.

Plot of error vs epochs




SGD Error vs Epochs
Error
Epochs
18000
17000
16000
15000
14000
13000
0
20000
40000
60000
80000
100000
Learning rate : 1e-05
Learning rate : 1e-06
Learning rate : 1e-07

Questions to ponder on:

1. **Do all three methods give the same/similar results? If yes, Why? Which method, out of the three would be most efficient while working with real world data?**

Ans: Yes, all three methods give similar results. With very small differences. This is because the dataset is relatively small and the number of features is also low.
For real world data, the most efficient method depends on 2 factors, the number of features/parameters and the size of the dataset. If the number of parameters is low, the normal equations method would be the most efficient as it would result in the most accurate model. If the number of parameters increase, the normal equations method no longer remains efficient as calculating the inverse and product of two matrices are expensive operations. In such a case the method chosen depends on the size of the dataset. If the size of the dataset is relatively small, Gradient descent method is the most efficient as it converges faster. If the dataset is very large, Stochastic GRadient Descent method will be the most efficient.

2. **How does normalization/standardization help in working with the data?**

Ans: The standardization method used by ud i.e: Min-max Standardization helps us bring all the features to a common scale. This is done to eliminate the effect that the range of any feature may have on our resulting model. For example, the feature age has values ranging between 18-64 whereas values of the feature children range between 0-5. These ranges are very different from each other and without any standardization, the value of age will influence the result much more than children due to its larger values. But this analysis is flawed as larger values don't necessarily mean more importance in the resulting model. After performing the Min-Max Standardization, values of all features are scaled to values in the range [0,1]. Hence, the variables can be compared on an equal footing now.

3. **Does increasing the number of training iterations affect the loss? What happens to the loss after a very large number of epochs (say, ~10^{10})**

Ans: Initially when the number of iterations is small, an increase in the number of iterations decreases the loss. As the number of iterations become larger and larger, the decrease in loss becomes smaller. After a very large number of epochs (~10^{10}), the loss stabilises somewhere close to the minimum loss and further iterations lead to a negligible decrease in loss.

4. **What primary difference did you find between the plots of Gradient Descent and Stochastic Gradient Descent?**

Ans: Gradient descent algorithm converges slowly, but once the convergence happens, the algorithm stays steadily at the point of minimum error without a lot of oscillations. It gives a very optimal solution to the linear regression problem. Stochastic gradient descent does not converge smoothly, and the solution given is not as optimal as gradient descent. Also, stochastic gradient descent has the tendency to oscillate away from the minima even after convergence.

5. **What would have happened if a very large value (2 or higher) were to be used for learning rate in GD/SGD?**

Ans: The learning rate determines by how much we are adjusting the weights of our features after each epoch of our algorithm. Thus, the learning rate determines how fast or slow we will reach the optimal weights. The higher the value of the learning rate the faster we travel down the slope. However, if the learning is too high (2 or higher) it will drastically update the weights of our features leading to divergent behavior. This will result in our model oscillating after each epoch. Larger weights induce large gradients which result in a large update being made to the weights and our model rapidly moves away from the minima.

6. **Would the minima (minimum error achieved) have changed if the bias term (w0) were not to be used, i.e. the prediction is given as Y=WTX instead of Y=WTX+B.**

Ans: Yes, the minimum error achieved when the bias term is not used would be greater than the minimum error achieved when bias term is used. If the bias term is not used, the modelled linear equation would be a line passing through origin. If the data does not represent such a linear equation, the minimum error achieved will be higher. In fact, modelling data with a linear regression model using bias would predict the value of w0 as 0 if the data is indeed represented by a linear equation passing through origin. However for this dataset, ince the bias term takes on a very small value, it has very little influence. Hence not using the bias term would only lead to a negligible increase in the minimum error achieved compared to the case where it is used.

7. **What does the weight vector after training signify? Which feature, according to you, has the maximum influence on the target value (insurance amount)? Which feature has the minimum influence?**

Ans: The weight vector after training are the coefficients of the features in the parametrized equation of our linear regression hypothesis. The higher the corresponding weight of any feature, the more influence it has on the target value.

The feature with the **Maximum** influence on the insurance amount is:	**BMI**
The feature with the **Minimum** influence on the insurance amount is:	**Children**