FODS Assignment-2

Rishab Nahar - 2018A7PS0173H
Sarvesh Khetan - 2018A4PS0947H
Samkit Jain - 2017B2A71723H

1.1 Introduction:

The aim of the project is to implement multi linear regression via three different methods namely:

A. normal equations
B. gradient descent
C. stochastic gradient descent

This assignment teaches us how actually the algorithms written in the sklearn library actually works and in depth mathematical intuition behind these algorithms.

The dataset used in this project consists of three independent features i.e. age(X1), bmi(X2) and number of children of an individual(X3) and a dependent feature insurance amount for that person(Y).Our goal is to use these independent features to predict the values of dependent feature for which we need to devise algorithms to learn the coefficients w.r.t. each independent variable.

$$Y = C + M_1X_1 + M_2X_2 + M_3X_3$$

How to run this project?

1. Install Anaconda/Jupyter Notebook
2. Install all dependencies/libraries as stated later
3. Download all the given files and put them in a folder named “X” on the desktop
4. open Anaconda Prompt and type <jupyter notebook>
5. Desktop -> X -> Linear Regression
6. Run the ipynb file

1.2 Code Design, Methodology and Derivations:

The code can be divided into following subsections for clear understanding:

A. Importing Required Libraries:

You need to install and import numpy, pandas and matplotlib libraries to successfully run this project

B. Defining Our Normal Equation Class:

Here once we get the error function we differentiate it w.r.t. all the weights and equate all the derivatives to zero because we want to reach the global minima of the curve and at the global minima slope is zero i.e. derivative of the function is zero. Hence we need to solve the n+1 simultaneous equation for n independent + 1 bias variable.

To understand the math and derivation of the vectorized formula used in this class please refer to the following pictures.It is better to vectorize our algorithm rather than using a loop to iterate over all the data points because it makes our algorithm more efficient.

(M1) Using normal equations.

x^1	x^2	....	x^n	y
x_1^1	x_1^2	...	x_1^n	y_1
.	.		.	.
.	.		.	.
x_m^1	x_m^2	...	x_m^n	y_m

⇒ n independent features.

⇒ m datapoints

$\therefore \hat{y} = c + m_1 x^1 + m_2 x^2 + \dots + m_n x^n$

$SS_{reg} = \sum_{i=1}^{m} (\hat{y} - y_i)^2$

$= \sum_{1}^{m} (c + m_1 x_i^1 + \dots m_n x_i^n - y_i)^2$

(N+1) equations

$\therefore \frac{\delta SS_{reg}}{\delta c} = \sum_{1}^{m} (c + m_1 x_i^1 + \dots m_n x_i^n - y_i)(1) = 0$

$\therefore \frac{\delta SS_{reg}}{\delta m_1} = \sum_{1}^{m} (c + m_1 x_i^1 + \dots m_n x_i^n - y_i)(x_i^1) = 0$

$\therefore \frac{\delta SS_{reg}}{\delta m_n} = \sum_{1}^{m} (c + m_1 x_i^1 + \dots m_n x_i^n - y_i)(x_i^m) = 0$

(N+1) equations:

$\therefore c\sum_1^m + m_1 \sum_1^m x_i^1 + \dots m_n \sum_1^m x_i^n = \sum_1^m y_i$

$\therefore c\sum x_i^1 + m_1 \sum x_i^1 x_i^1 + m_2 \sum x_i^1 x_i^2 \dots + m_n \sum x_i^n x_i^1 = \sum y_i x_i^1$

$\therefore c\sum x_i^2 + m_1 \sum x_i^1 x_i^2 + m_2 \sum x_i^2 x_i^2 \dots + m_n \sum x_i^n x_i^2 = \sum y_i x_i^2$

$\therefore c\sum x_i^n + m_1 \sum x_i^1 x_i^n + m_2 \sum x_i^2 x_i^n + \dots + m_n \sum x_i^n x_i^n = \sum y_i x_i^n$

Now representing same (N+1) equations in matrix form.

$$\begin{bmatrix} \Sigma 1 & \Sigma x_i^1 & \Sigma x_i^2 & \dots & \Sigma x_i^n \\ \Sigma x_i^1 & \Sigma x_i^1 x_i^1 & \Sigma x_i^2 x_i^1 & \dots & \Sigma x_i^n x_i^1 \\ \vdots & & & & \\ \Sigma x_i^n & \Sigma x_i^1 x_i^n & \Sigma x_i^2 x_i^n & \dots & \Sigma x_i^n x_i^n \end{bmatrix} \begin{bmatrix} c \\ m_1 \\ m_2 \\ \vdots \\ m_n \end{bmatrix} = \begin{bmatrix} \Sigma y_i \\ \Sigma y_i x_i^1 \\ \Sigma y_i x_i^2 \\ \Sigma y_i x_i^3 \\ \vdots \\ \Sigma y_i x_i^n \end{bmatrix}$$

yes, you are done here but you can instead simplify your calculations by using matrix instead of Σ as used above. Now let's see same solution as above in matrix form. (Some people call this vectorization, which basically means using matrix)

$y = c + m_1 x^1 + m_2 x^2 + \dots + m_n x^n$

(y: small)

now assume $[1\ x^1\ x^2 \dots x^m]$ as x (small) and $[c\ m_1\ m_2 \dots m_n]$ as θ.

$$\therefore y = [1\ x^1\ x^2 \dots x^m] \begin{bmatrix} c \\ m_1 \\ \vdots \\ m_n \end{bmatrix} = x \cdot \theta^T$$

$$\therefore SS_{res} = \sum_1^m (\hat{y} - y_i)^2 = \sum_1^m (c + m_1 x_i^1 + m_2 x_i^2 + \dots m_n x_i^n - y_i)^2$$

$$= (c + m_1 x_1^1 + m_2 x_1^2 + \dots m_n x_1^n - y_1)^2 +$$
$$(c + m_1 x_2^1 + m_2 x_2^2 + \dots m_n x_2^n - y_2)^2 +$$
$$\vdots$$
$$(c + m_1 x_m^1 + m_2 x_m^2 + \dots m_n x_m^n - y_m)^2$$

Now lets consider following table.

→ reason why we put an extra column of 1 in preprocessing step.

x^0	x^1	x^2	...	x^n	y
1	x_1^1	x_1^2	...	x_1^n	y_1
1	:	:		:	:
:	:	:		:	:
1	x_m^1	x_m^2	...	x_m^n	y_m

$Y_{1\times m} = [y_1\ y_2 \ldots\ y_m]$ (capital)

$X_{m\times(n+1)} = \begin{bmatrix} 1 & x_1^1 & x_1^2 & \ldots & x_1^n \\ 1 & : & : & & : \\ : & : & : & & : \\ 1 & x_m^1 & x_m^2 & \ldots & x_m^n \end{bmatrix}$ (capital)

$$SS_{res} = (X\theta^T - Y^T)^T \cdot (X\theta^T - Y^T)$$

$$= \left(\begin{bmatrix} 1 & x_1^1 & \ldots & x_1^n \\ : & : & & \\ 1 & x_m^1 & \ldots & x_m^n \end{bmatrix}\begin{bmatrix} c \\ : \\ m_n \end{bmatrix} - \begin{bmatrix} y_1 \\ : \\ y_m \end{bmatrix}\right)^T \cdot \left(\begin{bmatrix} 1 & x_1^1 & \ldots & x_1^n \\ : & : & & \\ 1 & x_m^1 & \ldots & x_m^n \end{bmatrix}\begin{bmatrix} c \\ : \\ m_n \end{bmatrix} - \begin{bmatrix} y_1 \\ : \\ y_m \end{bmatrix}\right)$$

$$= \underbrace{\begin{bmatrix} c+m_1x_1^1+\ldots m_nx_1^n-y_1 \\ : \\ c+m_1x_m^1+\ldots m_nx_m^n-y_m \end{bmatrix}^T}_{1\times m} \cdot \underbrace{\begin{bmatrix} c+m_1x_1^1+\ldots m_nx_1^n-y_1 \\ : \\ c+m_1x_m^1+\ldots m_nx_m^n-y_m \end{bmatrix}}_{m\times 1}$$

$$= (c+m_1x_1^1+\ldots m_nx_1^n-y_1)^2 + (c+m_1x_2^1+\ldots m_nx_2^n-y_2)^2 + \ldots + (c+m_1x_m^1+m_2x_m^2+\ldots m_nx_m^n-y_m)^2$$

Hence we got same SS_{res} value, means our matrix formula for SS_{res} is correct.

$$\therefore SS_{res} = (X\theta^T - Y^T)^T \cdot (X\theta^T - Y^T)$$
$$= (X\theta^T - Y^T) \cdot (X\theta^T - Y^T)^T$$
$$[\because a \cdot b = b \cdot a]$$

$$= (X\theta^T - Y^T) \cdot (X^T\theta - Y)$$
$$= XX^T\theta\theta^T - X\theta^T \cdot Y - Y^T \cdot X^T\theta + YY^T$$

put.

now lets assume $X\theta^T \cdot Y = k$

$$\Rightarrow Y^T X^T \theta = k^T$$

dimension of $X \rightarrow (m) \times (n+1)$
dimension of $\theta^T \rightarrow (n+1) \times (1)$
dimension of $Y \rightarrow (1) \times (m)$
Hence dimension of $K \rightarrow (m)(m)$
Hence K is a square matrix, similarly you can also prove that K is a symmetric matrix.

$$\therefore \boxed{K^T + K = 2K} \quad = 2K^T$$

$$\therefore SS_{res} = XX^T\theta\theta^T - (2)(X\theta^T Y) + YY^T$$
$$= XX^T\theta\theta^T - (2)(X^T\theta Y^T) + YY^T$$

Now we need to solve $\frac{\delta SS_{res}}{\delta c} = 0$, $\frac{\delta SS_{res}}{\delta m_1} = 0$, ... $\frac{\delta SS_{res}}{\delta m_n} = 0$. All these equations can be written together as in matrix form $\frac{\delta SS_{res}}{\delta \theta} = 0$

$$\therefore \frac{\delta SS_{res}}{\delta \theta} = (XX^T\theta)(1) + (XX^T\theta^T)(1) - (2)(X^T Y^T)$$
$$\frac{\delta}{\delta \theta} = (2)(XX^T\theta) - (2)(X^T Y^T) = (2)(XX^T\theta^T) - (2)(X^T Y^T)$$

$$\therefore (2)(X^T Y^T) = (2)(XX^T\theta)$$
$$\therefore \boxed{\theta = (X^T \cdot X)^{-1} \cdot (X^T \cdot (Y^T))}$$

→ or just Y also gives same ans.

As you can see that since this method involves inversion of a matrix and inverting a matrix is computationally very expensive hence we required a more efficient method to reach the global minima and one of the way is gradient descent.One is advised to use this method if no of independent variables is 1000 or less, though this is not a strict condition but can be handy.(source-- Andrew Ng)

C. Defining Our Gradient Descent Class:

In this method we don't directly go to the minima but instead we reach close to the global minima of the cost/error function gradually by jumping from point to another point.This new point in every iteration is decided by the gradient at that old point.

Initial / starting value of these weights can be taken any random value, but we have taken them as all zeros, it won't matter at what initial point one is starting because ultimately the goal is to converge to the global minima.

The weights are updated using following formula:

$$(M_i)new = (M_i)old - (\alpha)*((\partial E \,/\, (M_i)old))$$

Where,

M = weight/coefficient α = learning rate

i = 0,1,.....no of weights

E = sum of squared error function = $1/2 * \sum_1^n (y - y')^2$

Y' represents predicted values while Y represents actual target values for n data points.

Vectorization of the algorithm and respective derivation of formula can be seen below:

since initially you need to take any random value for $c, m_1, \ldots m_n$ best is to choose all of them as "0".

m2) Gradient Descent

$$\therefore SS_{res} = \sum_{1}^{m} (\hat{y} - y_i)^2$$

$$= (X\theta^T - y^T) \cdot (X\theta^T - y^T)$$

(proved earlier)

$$\therefore \boxed{SS_{res} = (X\theta^T - y^T) \cdot (X\theta^T - y^T)^T}$$

Now we need to find

$$(c)_{new} = (c)_{current} - (\alpha) \left(\frac{\delta SS_{res}}{\delta c}\right)_{current}$$

$$(m_1)_{new} = (m_1)_{current} - (\alpha) \left(\frac{\delta SS_{res}}{\delta m_1}\right)_{current}$$

$$\vdots$$

$$(m_n)_{new} = (m_n)_{current} - (\alpha) \left(\frac{\delta SS_{res}}{\delta m_n}\right)_{current}$$

All these above equations can be represented in matrix form as,

$$\begin{bmatrix} c_{new} \\ (m_1)_{new} \\ \vdots \\ (m_n)_{new} \end{bmatrix} = \begin{bmatrix} c_{current} \\ (m_1)_{current} \\ \vdots \\ (m_n)_{current} \end{bmatrix} - (\alpha) \begin{bmatrix} \delta SS_{res}/\delta c \\ \delta SS_{res}/\delta m_1 \\ \vdots \\ \delta SS_{res}/\delta m_n \end{bmatrix}_{current}$$

$$\therefore \theta_{new} = \theta_{current} - (\alpha) \left(\frac{\delta SS_{res}}{\delta \theta}\right)_{current}$$

Now we derived earlier that

$$\delta SS_{res}/\delta\theta = (2)(X^T X\theta - X^T y^T)$$

$$\therefore \boxed{\theta_{new} = \theta_{current} - (2)(\alpha)(X^T) \cdot (X\theta_{current} - y^T)}$$

or $\theta^T_{current}$ gives same ans.

or y gives same ans

The stopping criteria for our implementation is when change in error difference is less than 0.00000001 but if no of iterations specified by the user is less than the no of iterations taken to reach error difference less than 0.00000001 then the algorithm stops when the no of iterations is equal to the one defined by the user.

As you can see that since this method involves calculating error w.r.t all the datapoints, we can further implore the time taken to run the algorithm by considering only few data points, this is called mini batch gradient descent and if only 1 datapoint is considered then it is called stochastic gradient descent.One is advised to use this method if no of independent variables is 1000000 or less, though this is not a strict condition but can be handy.(source-- Andrew Ng)

D. Defining Our Stochastic Gradient Descent Class:

This algorithm is almost exactly the same as gradient descent but the only difference here is that while considering the error function in gradient descent we used error due to all points but in stochastic gradient descent we consider error only due to a single random point in the dataset.

Hence now the weights are updated using following formula:

$$(M_i)new = (M_i)old - (\alpha) * ((\partial E \ / \ (M_i)old))$$

Where,

M = weight/coefficient $\alpha = \text{learning rate}$

i = 0,1,.....no of weights

E = sum of squared error function = $1/2 * (y_p - y_p')^2$where $y_p$and y_p' are actual target value of that chosen data point and predicted value of that data point respectively

The derivations for the vectorised formula can be understood from following image:

start with
$\theta_{current} = [0, 0, 0 \ldots 0]$

(m3) Stochastic Gradient Descent.

$\therefore SSres = (\hat{y} - y_i)^2$ | at random point $x_p^1, x_p^2, x_p^3 \ldots x_p^n, y_p$.

$= (c + m_1 x_p^1 + \ldots m_n x_p^n - y_p)^2$

Now assume this table.

x^0	x^1	x^2	.....	x^n	y
1	x_1^1	x_1^2	.....	x_1^n	y_1
1	x_2^1				
1	x_p^1	x_p^2		x_p^n	y_p
1	x_m^1	x_m^2	.....	x_m^n	y_m

(same as prev) — x — (same as prev) — y — capital — $y_p = [y_p]$

$X_p = [1 \quad x_p^1 \quad x_p^2 \ldots x_p^n]$ (capital)

$\therefore (c + m_1 x_p^1 + \ldots m_n x_p^n - y_p) = [1 \; x_p^1 \ldots x_p^n] \begin{bmatrix} c \\ m_1 \\ \vdots \\ m_n \end{bmatrix} - [y_p]$

$= (X_p)(\theta^T) - y_p^T$

$\therefore SSres = (X_p\theta^T - y_p^T)^T \cdot (X_p\theta^T - y_p^T)$

$SSres = (X_p\theta^T - y_p^T) \cdot (X_p\theta^T - y_p^T)^T$ → used in actual algo.

By doing same calculations you can get

$\theta_{new} = \theta_{current} - (\alpha)(2)(X_p^T) \cdot (X_p \cdot \theta_{current} - y_p^T)$

→ used in actual algo.

E. Defining Our Evaluation Metric Class:

This class is used to calculate the accuracy of our model.Now we know that there are various accuracy measures but here we have implemented three of them namely RMSE(root mean squared error) , MSE(mean squared error) and Total Error.Formula for each of these are given below.

RMSE = $\sqrt{(\sum_1^n \quad \left(y - \underline{y}\right)^2)/n}$

MSE = $(\sum_1^n \quad \left(y - \underline{y}\right)^2)/n$

Total Error(SSRES) = $0.5 * \sum_1^n \quad \left(y - \underline{y}\right)^2$

For obvious reasons the lower the error i.e. closer to 0 the better is our model.All of these evaluation metrics are proportional to each other and hence can be used based on one's will.

F. Data Preprocessing:

Since the given data is not already in normal form you need to normalize/standardize the entire dataset using

X -> (X - mean)/(standard deviation)

You also need to add a bias column i.e. a column of all 1s because as shown in the derivation of formula.

Then a train test split of 70:30 was performed.

G. Data Learning using Normal Equation Method:

Results when no shuffling was performed

Total Error (Training) = `401.63598553252405`

Total Error (Testing) = `188.19593504484376`

Coefficients/Weights = `[-0.02981352 0.2999276 0.15798607 0.03405244]`

Results when 20 shuffling was performed

Mean Total Error (Training) = 407.547979482744
Variance Total Error (Training) = 127.56772122629414

Mean Total Error (Testing) = 181.9147928815619
Variance Total Error (Testing) = 128.42527184323646

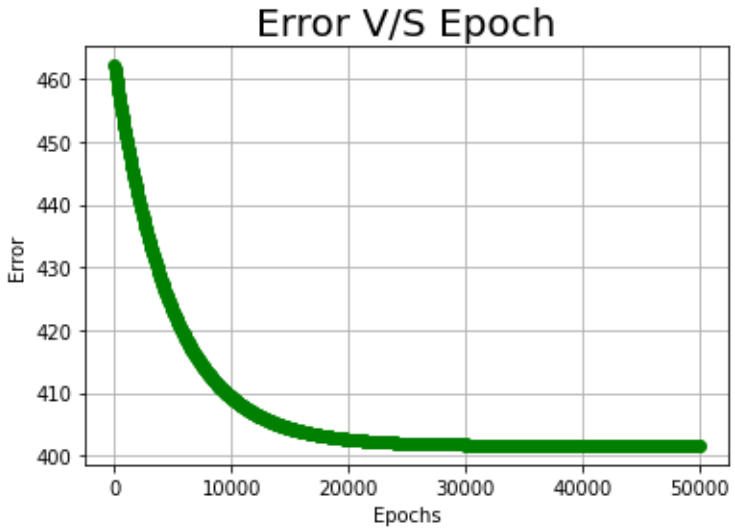
Avg Coefficients/Weights = [-0.003149987945818418, 0.2827575060018899, 0.16531337203091967, 0.05297938776660495]

H. Data Learning using Gradient Descent Method:

Results when no shuffling was performed

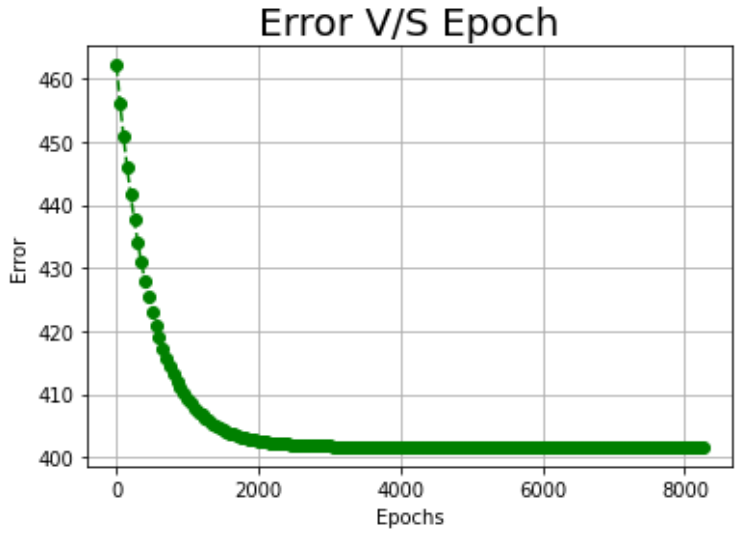
For Learning Rate = 10^{-07}

Total Error(Training) = 401.6385170541879
Total Error (Testing) = 188.10701551129267
Coefficients/Weights = [-0.02894347 0.29787353 0.1574870 0.03429231]



For Learning Rate = 10^{-06}

Total Error(Training) = 401.6359909923883
Total Error (Testing) = 188.19067399116233
Coefficients/Weights = [-0.02975871 0.29983663 0.1579846 0.03407975]

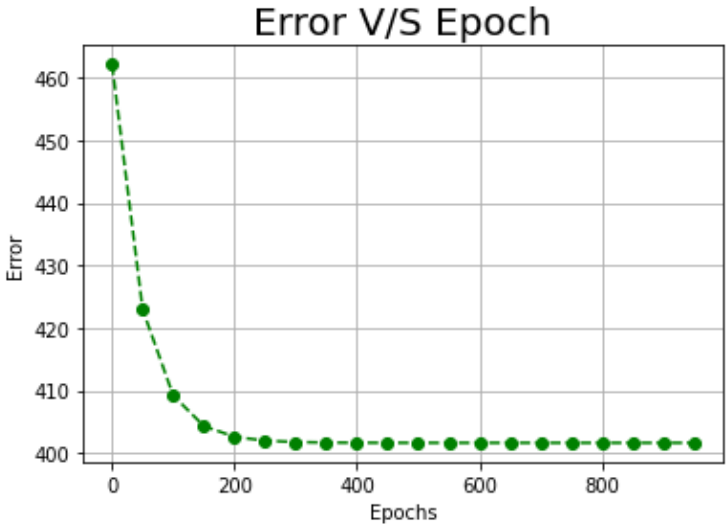


For Learning Rate = 10^{-05}

Total Error(Training) = `401.63598607179506`

Total Error (Testing) = `188.19420098611772`

Coefficients/Weights = `[-0.02979538 0.29989989 0.1579881 0.0340627]`



As we can conclude from the following graphs that as the learning rate decreases the no of iterations (epochs) taken to reach the minima increases thus training time increases.

As we can observe that training error and testing error achieved for different learning rates are close to what was achieved using normal equations, this ensures that our gradient descent algorithm is

converging close to the global minima of the cost/error function cause in normal equation what we calculated is error exactly at the global minima.

The minimum training and testing error differs slightly from each other in each case because as we know we are not reaching exact global minima we reach very close to the global minima and hence at different times we might reach different close to the global minima.

Results when 20 shuffling was performed

Mean Total Error (Training) = `407.547979482744`
Variance Total Error (Training) = `127.56772122629414`

Mean Total Error (Testing) = `181.9147928815619`
Variance Total Error (Testing) = `128.42527184323646`

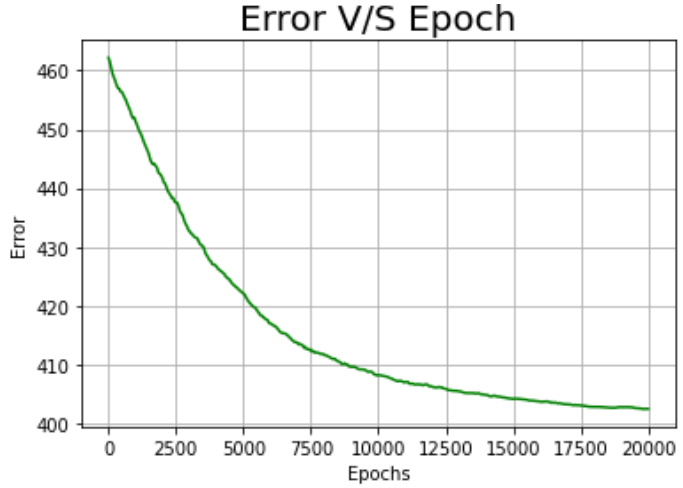
Avg Coefficients/Weights = `[-0.003149987945818418, 0.2827575060018899, 0.16531337203091967, 0.05297938776660495]`

I. Data Learning using Stochastic Gradient Descent Method:

Results when no shuffling was performed

For Learning Rate = 0.0001

Total Error(Training) = `402.4594101035025`
Total Error (Testing) = `187.68017306270383`
Coefficients/Weights = `[-0.03688233 0.26246389 0.14765728 0.04254964]`

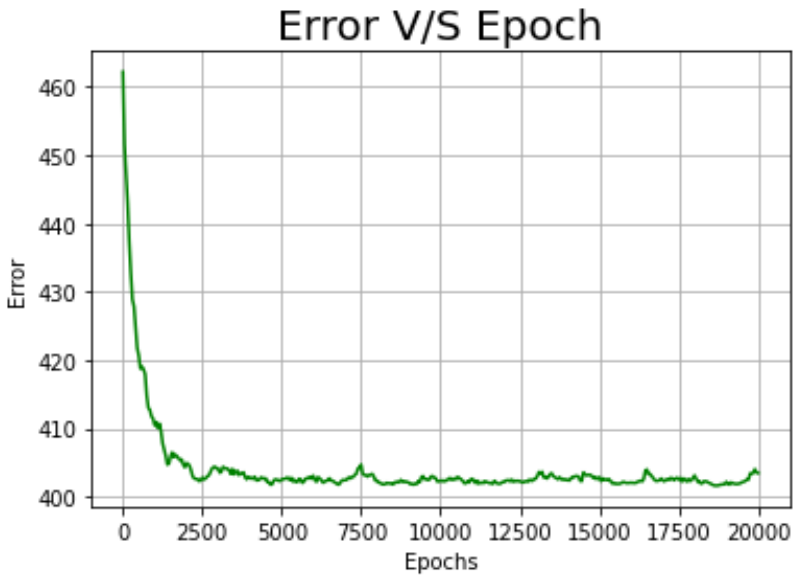


For Learning Rate = 0.001

Total Error(Training) = 403.4853823989615

Total Error (Testing) = 186.22568335410477

Coefficients/Weights = [0.01506198 0.28206708 0.19941711 0.03679804]

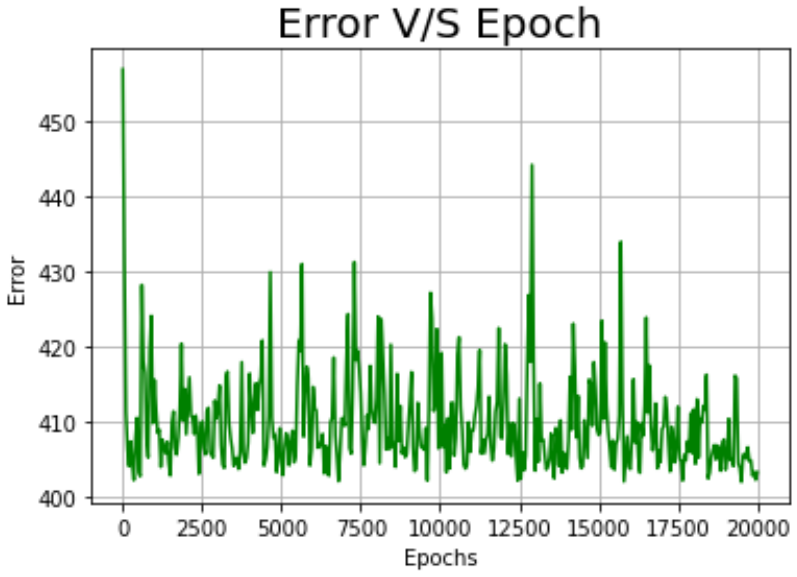


For Learning Rate = 0.01

Total Error(Training) = 403.2684255808458

Total Error (Testing) = 187.33013370490636

Coefficients/Weights = [-0.00901945 0.31408177 0.14356575 0.08718604]



Results when 20 shuffling was performed

Mean Total Error (Training) = 411.85803265120694
Variance Total Error (Training) = 148.02488043436006

Mean Total Error (Testing) = 179.35172464904977
Variance Total Error (Testing) = 156.22989375600605

Avg Coefficients/Weights = [0.002120855261627242, 0.2777814374162087, 0.17030334973652012, 0.05128311149474115]

1.3 Point to Ponder:

Do all three methods give the same/similar results? If yes, Why?
Yes because ultimately we are wanting to converge to the same global minima just the way to converge has changes in fact in the normal equation we are not converging to the minima we are directing going to the minima.

Which method, out of the three would be most efficient while working with real world data?
Considering real world data is big data stochastic gradient descent will work the best because it will give us outputs in less time compared to other methods and is a computationally efficient method compared to other methods.

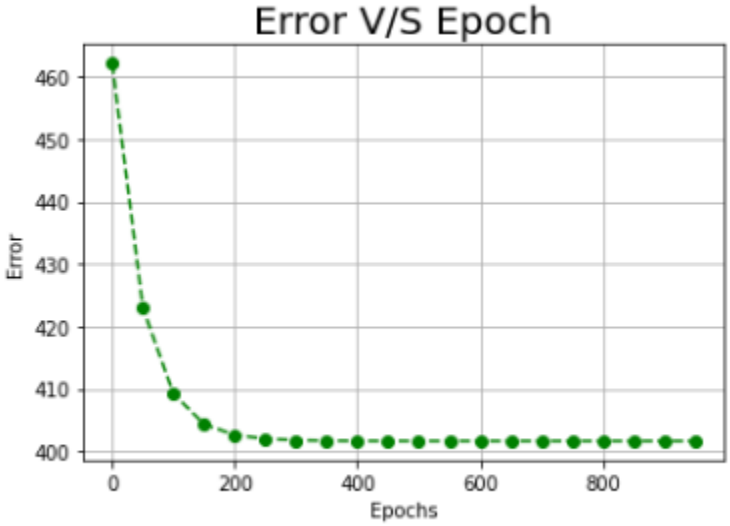
How does normalization/standardization help in working with the data?
Normalization helps to convert all the features to the same scale.But what is the issue with different scales?the issue is that say is feature 1 is of scale 1000 and feature 2 is of scale 0.1 then due to feature 1 you training time will increase.Thus if we scale feature 1 to same scale of feature 2 it will improve our training time.Now as a standard procedure we scale all features to values between -1 and 1 and best way to do this is converting distribution of that feature into a normal distribution, this is called normalization.Thus normalization helps in reducing the time taken by the algorithm to converge(training time).

Does increasing the number of training iterations affect the loss?
Yes, however it is significant only in initial stages because when no of iterations are too large the reduction in error becomes negligible.This happens because ultimately we are going to reach close to the same global minima but after a certain amount of iterations we would have already reached very close to the global minima and increasing the no of iterations further would not have much effect cause slope near the global minima is tending to 0.

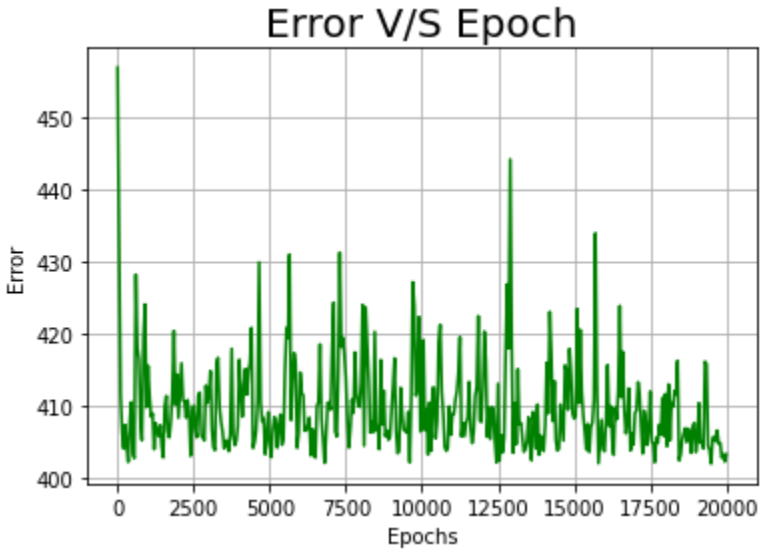
What happens to the loss after a very large number of epochs (say, ~1010)
The change in loss will become negligible but not zero exactly because you cannot reach the exact global minima you always reach close to the global minima.

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

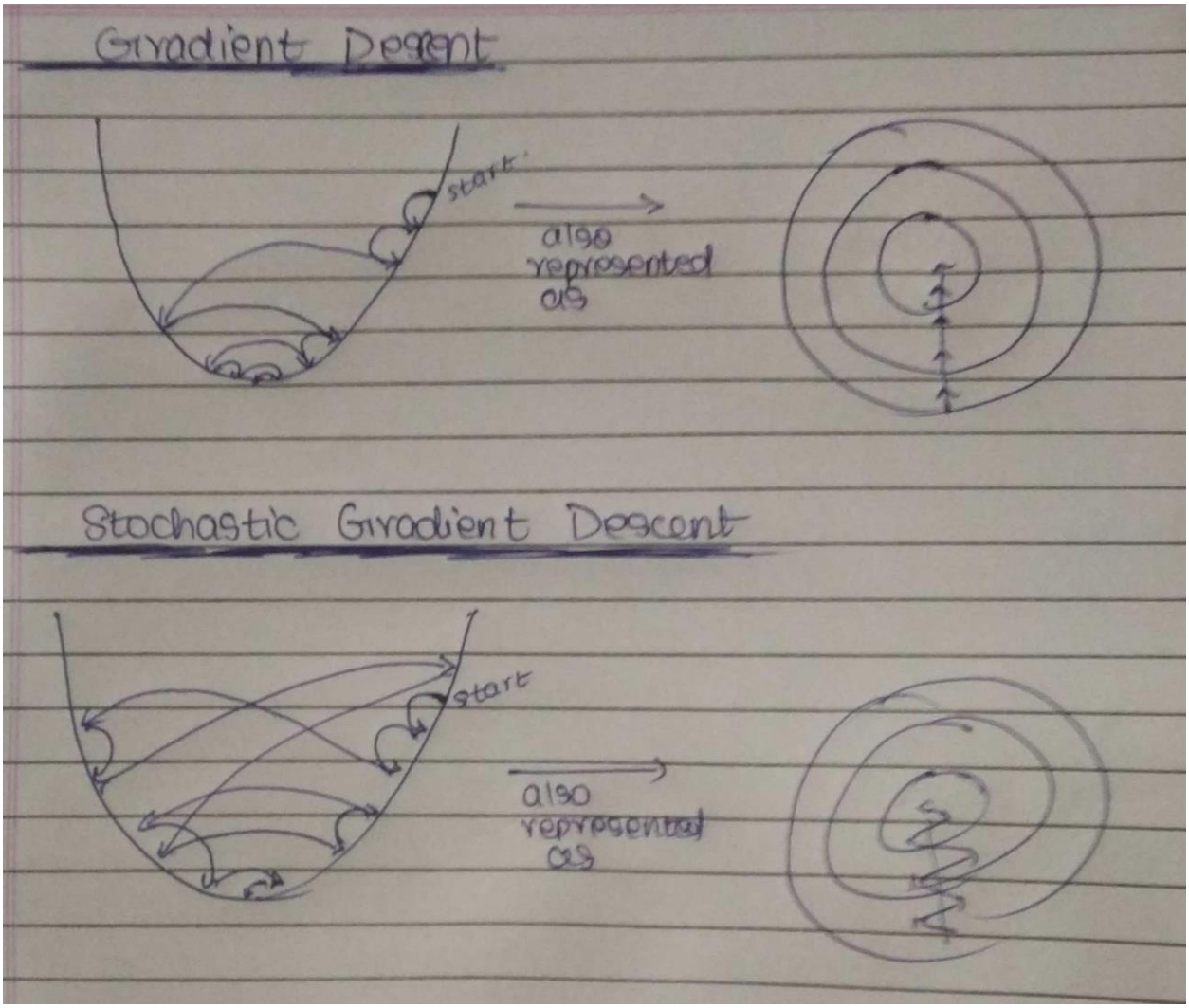
GD Plot



SGD Plot



As you can clearly see that both gd and sgd does converge to the global minima but the convergence of gd is smooth i.e. the error always decreases while in sgd the convergence is not smooth i.e. the error might increase sometimes in middle but overall it decreases and ensures that it reaches the global minima gradually.This is due to the fact that in sgd we consider only one randomly selected data point to calculate error.You can refer to follow diagram to understand what exactly is happening in gd and sgd.



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

It might have been possible that we did not converge to the global minima instead we might have diverged from the global minima.

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 □ = □□□ instead of □ = □□□ + □.

Not much in this case because the coefficient of bias term is approx -0.002 which is negligible compared to other independent features coefficients but it might be in other scenarios like consider an example where actual target variable is non zero value while values of all independent features is zero in such a case if we did not use bias term then the predicted value will be zero causing a error while if we used bias then predicted value will be non zero thus reducing the error compared to non bias.Bias variable can be thought of as used for shifting the regression function.

What does the weight vector after training signify?
The weight vector signifies the coefficients or the importance of each independent feature to predict the dependent(target) feature or in other words how much each feature contributes to the prediction of target/dependent variable.

Which feature, according to you, has the maximum influence on the target value (insurance amount)? Which feature has the minimum influence? The feature with highest coefficient/weight has the maximum importance which in this case is " age" while the feature with minimum coefficient/weight has the minimum importance which in this case is "number of children ".