Aim of this project / Problem Statement:

This project teaches us how actually the multiple linear regression algorithm is used to solve the regression problem and how it is written in the sklearn library and the indepth mathematical intuition behind this algorithm because I have implemented this algorithm from scratch.

The entire theory for the implementation can be understood by referring to my notes at following link (recommended as must read) : https://drive.google.com/drive/folders/1nhDkzNrkslPcaqFpiOUVKdgq_jpHFNvG?usp=sharing

The dataset used can be found at this link : https://drive.google.com/file/d/1Hnpbhnl0NU1-o4UJao2GukOZeQWiX6Gk/view?usp=sharing

By reading the theory you might have understood that ultimately the algorithm reduces to solving an unconstrained nonlinear optimization problem and we know there are several ways to solve this optimization problem so here we will be solve this optimization problem via these methods from scratch:

1. Normal Equations
2. Gradient Descent
3. Stochastic Gradient Descent

You can find the code for all of these at following link : https://drive.google.com/file/d/16AhDb5zLYxowEmUMTyHsmupHjUT_3x6Q/view?usp=sharing (open with google colab)

Code Design

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

A. Importing Required Libraries:

You need to import some essential basic manipulation libraries to do some math operations and hence we are importing these libraries here. The comments in the code specify exactly what purpose the library serves.

B. Data Preprocessing

1. First we use pandas library to read the dataset from the location where it has been uploaded.

   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.

2. Since the given dataset contains features with different scales hence we need to perform feature scaling i.e. we need to normalize/ standardize the entire dataset using

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

C. Data Learning Without Using SKLearn Library

1. 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.

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

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

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

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.

2. Using Normal Equations (Vectorized Formula) as Optimization Method:

Here first I have created a class which performs this optimization for us. The name of the class is 'NormalEquation'.This class contains several functions whose purpose is clearly mentioned in the code.

To understand the math and derivation of the vectorized formula used in this class the reader must go through the notes linked above. Once the reader has understood the mathematical derivation for the vectorized formula for solving multiple linear regression using the normal equation provided in the notes the reader will take no time to understand the code with the additional help provided via comments in the code wherever necessary.

The object of class ‘NormalEquation’ and ‘Evaluation Metric’ is created to use both these classes.

Results when no shuffling was performed

Now I have performed a 70:30 train test split to form the training and the testing dataset and applied the above devised algorithm to get following results

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

Now I have performed 20 times 70:30 train test split instead of doing just once which was done above to form the training and the testing dataset and applied the above devised algorithm to each split and then averaged the results to get following average values

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]`

3. Using Gradient Descent (Vectorized Formula) as Optimization Algorithm:

Here first I have created a class which performs this optimization for us. The name of the class is ‘GradientDescent’.This class contains several functions whose purpose is clearly mentioned in the code.

To understand the math and derivation of the vectorized formula used in this class the reader must go through the notes linked above. Once the reader has understood the mathematical derivation for the vectorized formula for solving multiple linear regression using the gradient descent provided in the notes the reader will take no time to understand the code with the additional help provided via comments in the code wherever necessary.

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.

As we know gradient descent can be implemented with different types of stopping criteria but here the stopping criteria for this implementation that I have taken 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.

The object of class ‘NormalEquation’ and ‘Evaluation Metric’ is created to use both these classes.

Results when no shuffling was performed

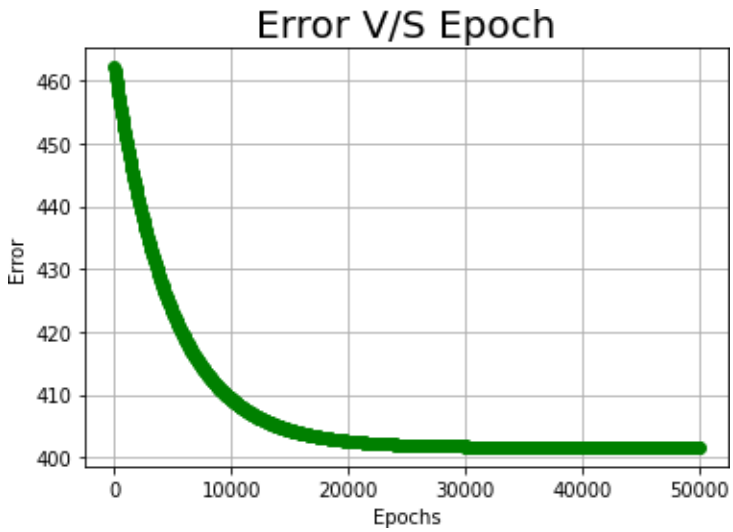
Now I have performed a 70:30 train test split to form the training and the testing dataset and applied the above devised algorithm to get following results.Now as we know learning rate is a hyperparameter I am doing hyperparameter tuning by trying out 3 different learning rates

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]`

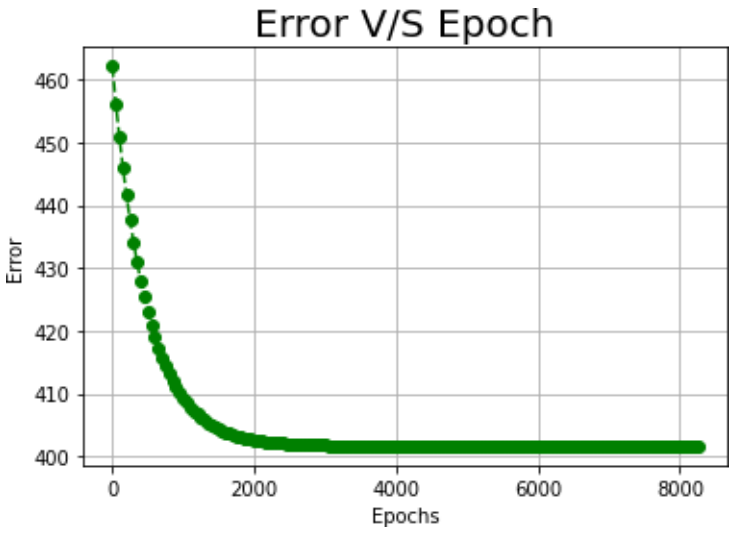


For Learning Rate = 10^{-06}

Total Error(Training) = `401.6359909923883`

Total Error (Testing) = `188.19067399116233`

Coefficients/Weights = `[-0.02975871 0.29983663 0.1579846 0.03407975]`

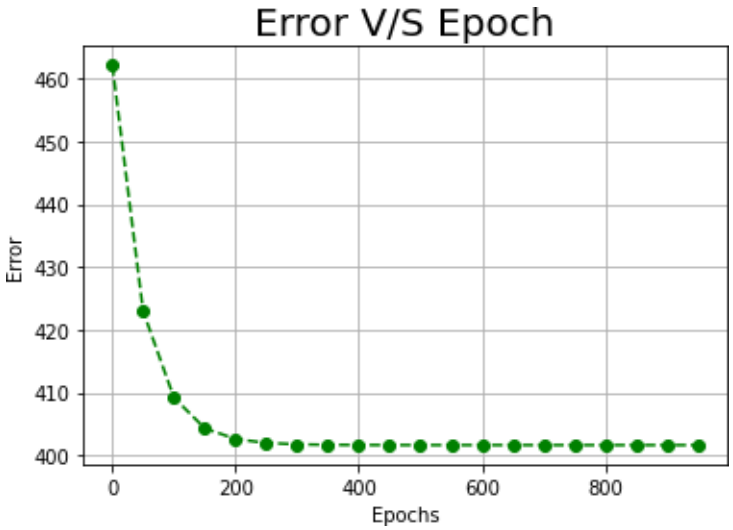


For Learning Rate = 10^{-05}

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 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.

Hence the best learning rate is 10^(-5) and hence using this for next steps

Results when 20 shuffling was performed

Now I have performed 20 times 70:30 train test split instead of doing just once which was done above to form the training and the testing dataset and applied the above devised algorithm to each split and then averaged the results to get following average values.

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]`

4. Using Stochastic Gradient Descent (Vectorized Formula) as Optimization Algorithm:

Here first I have created a class which performs this optimization for us. The name of the class is 'StochasticGradientDescent'.This class contains several functions whose purpose is clearly mentioned in the code.

To understand the math and derivation of the vectorized formula used in this class the reader must go through the notes linked above. Once the reader has understood the mathematical derivation for the vectorized formula for solving multiple linear regression using the stochastic gradient descent provided in the notes the reader will take no time to understand the code with the additional help provided via comments in the code wherever necessary.

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.

Here I have taken no of iterations specified by the user as the stopping criteria.

Results when no shuffling was performed

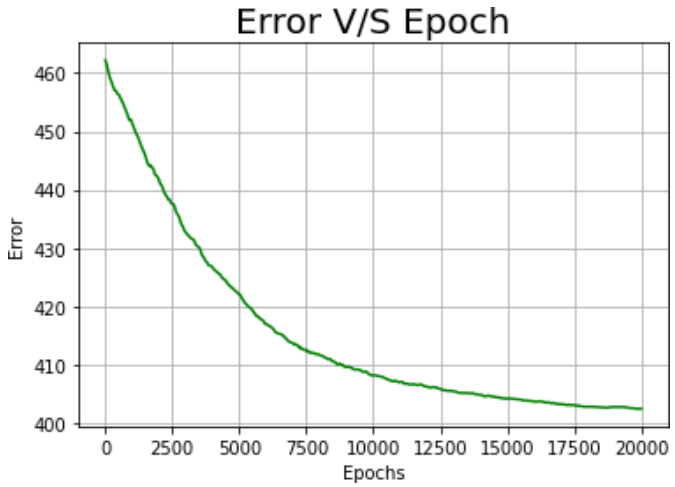
Now I have performed a 70:30 train test split to form the training and the testing dataset and applied the above devised algorithm to get following results.Now as we know learning rate is a hyperparameter I am doing hyperparameter tuning by trying out 3 different learning rates

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]`

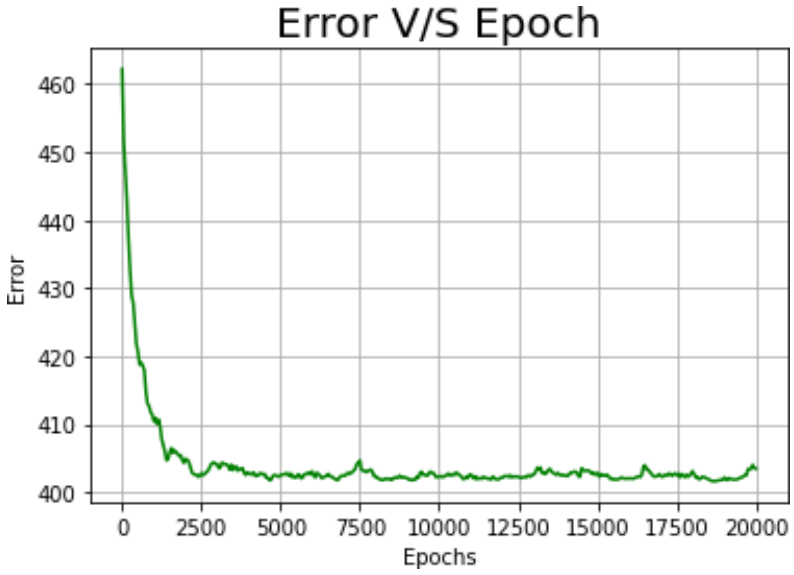


For Learning Rate = 0.001

Total Error(Training) = `403.4853823989615`

Total Error (Testing) = `186.22568335410477`

Coefficients/Weights = `[0.01506198 0.28206708 0.19941711 0.03679804]`

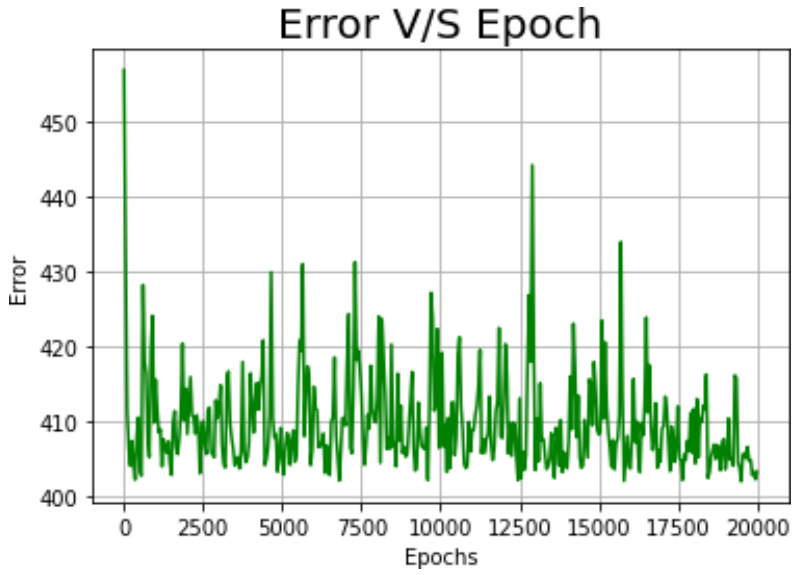


For Learning Rate = 0.01

Total Error(Training) = 403.2684255808458

Total Error (Testing) = 187.33013370490636

Coefficients/Weights = [-0.00901945 0.31408177 0.14356575 0.087186



Results when 20 shuffling was performed

Now I have performed 20 times 70:30 train test split instead of doing just once which was done above to form the training and the testing dataset and applied the above devised algorithm to each split and then averaged the results to get following average values.

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]`

D. Data Learning Using SKLearn Library

Firstly a 70:30 train test split has been performed. Then an object is created of the linear regression class already defined in SKLearn Library.Then the train dataset is fitted using this object and following weights are learnt.

```
linear model coeff (w): [0.2999276  0.15798607 0.03405244]
linear model intercept (b): -0.029813521392349177

-----------------training evaluation metrics--------------------
rmse = 0.9459229722467289
r2_score = 0.11770834190185442

-----------------testing evaluation metrics----------------------
rmse = 0.920709734555855
r2_score = 0.12342534780862413
```

Hence we can see that these values are very close to the one that I got via my implementation and this confirms that what I have implemented is absolutely correct.