A Project Report

On

Building Linear Regression Using 3 Different Methods

BY

Kshitij Verma (2017B1A71145H)

Rahul Prakash (2018AAPS0893H)

Siddharth Raj (2018AAPS0398H)

Under the supervision of

Dr. NL Bhanu Murthy

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(OCTOBER , 2020)

Assignment 2: Multiple Linear Regression

Introduction: In statistics, linear regression is a linear approach to modeling the relationship between a scalar response and one or more explanatory variables. When more than one explanatory variables are involved, the process is called multiple linear regression.

We were given a dataset called "Insurance.txt". The file consisted of 3 independent variables, namely age, bmi and children. The one dependent variable is charges. We were asked to build a multi variable regression model using the dataset and predict the charges. 20 linear regression models are made using 3 different methods and error associated with each algorithm is calculated.

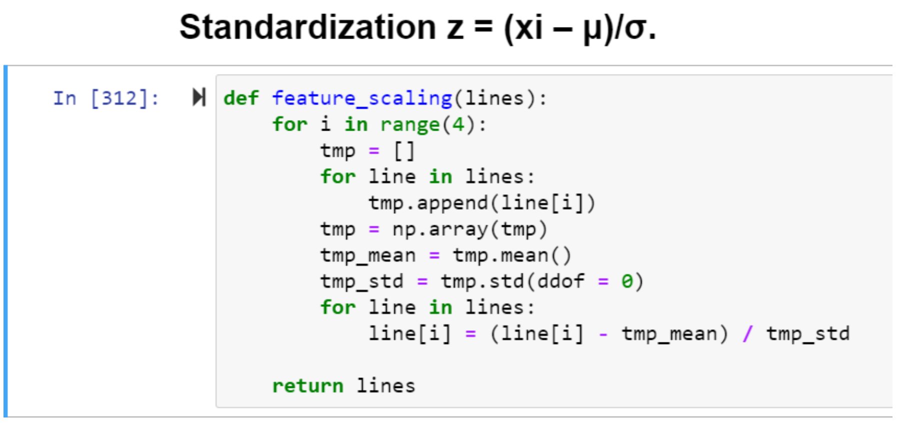
Steps in building the model:

Mentioned below are the steps involved in building the Model. They are:

1. **Data Pre-Processing:** Data Preprocessing is a technique that transforms the given raw data into a clean dataset. This step improves the accuracy of the model. Mostly 2 methods are used for data pre-processing. They are a) Standardization and b) Normalization.
   Normalization: This step is to make all the elements lie between 0 and 1 thus bringing all the values of numeric columns in the dataset to a common scale.
   Standardization: Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

   The formula to standardize the data is: $Z = \frac{(xi - \mu)}{\sigma}$

   Standardization is implemented in our code as :



Standardization z = (xi – μ)/σ.

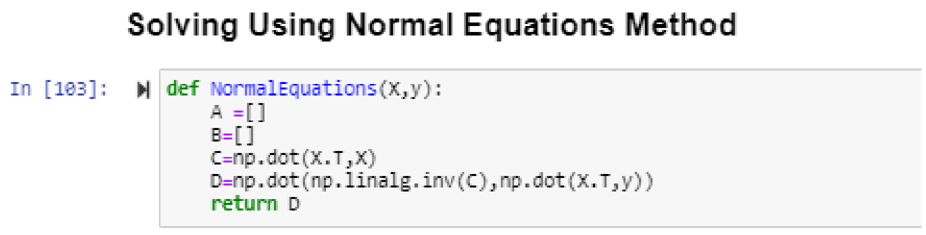
```
In [312]:  def feature_scaling(lines):
               for i in range(4):
                   tmp = []
                   for line in lines:
                       tmp.append(line[i])
                   tmp = np.array(tmp)
                   tmp_mean = tmp.mean()
                   tmp_std = tmp.std(ddof = 0)
                   for line in lines:
                       line[i] = (line[i] - tmp_mean) / tmp_std

               return lines
```

2. **Building the model:** The data was read from "Insurance.txt" file. Random shuffling of the data is done each time a regression model is made using the random shuffle function. After random shuffling, the entire dataset is divided into two parts, training set and testing set. The training set consists of 70% of the dataset and testing set consists of remaining 30% of the dataset. The 3 different algorithms, the Normal Equations method, the gradient descent method and Stochastic gradient descent method is run for each of the 20 regression models and errors are calculated.

1. **Using Normal Equations Method:** The formula used to calculate the Weights using normal equations method is :

$$\Theta = (X^T X)^{-1}.(X^T y)$$



Solving Using Normal Equations Method

```
In [103]:  def NormalEquations(X,y):
               A =[]
               B=[]
               C=np.dot(X.T,X)
               D=np.dot(np.linalg.inv(C),np.dot(X.T,y))
               return D
```

2. **Using Gradient Descent Method:** The hypothesis function in the model is:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

Here, theta are the weights, x are the independent variable and h(x) is the predicted value. The hypothesis is calculated in the form of matrix multiplication using:

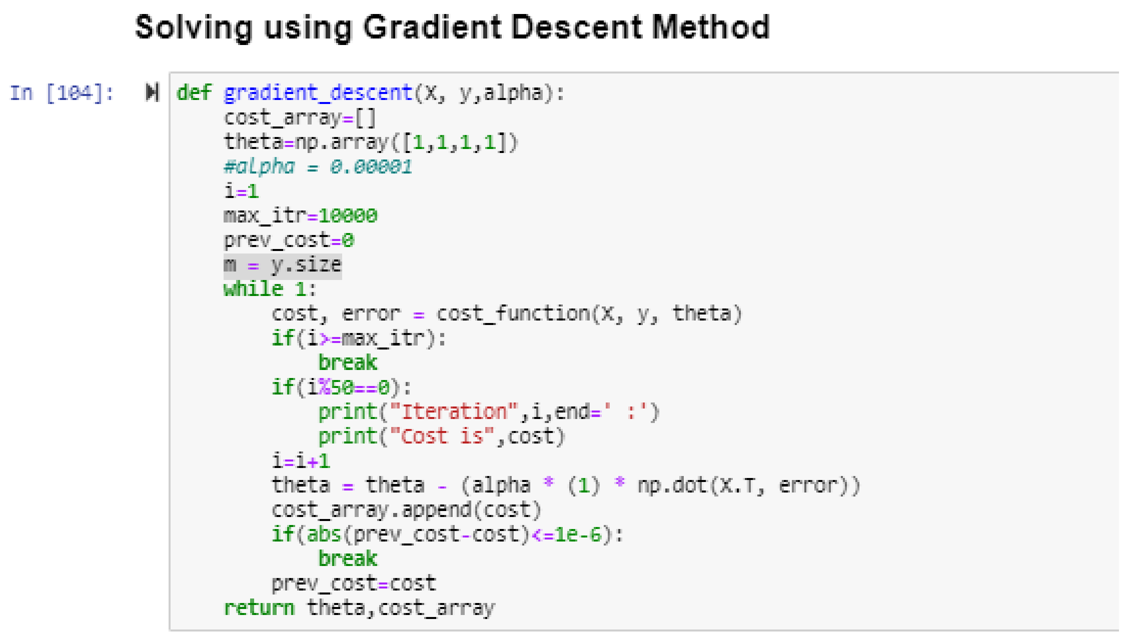
$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \ldots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

The Cost function is calculated as :

$$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y})$$

The weights after each iteration is calculated as :

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0..n}$$



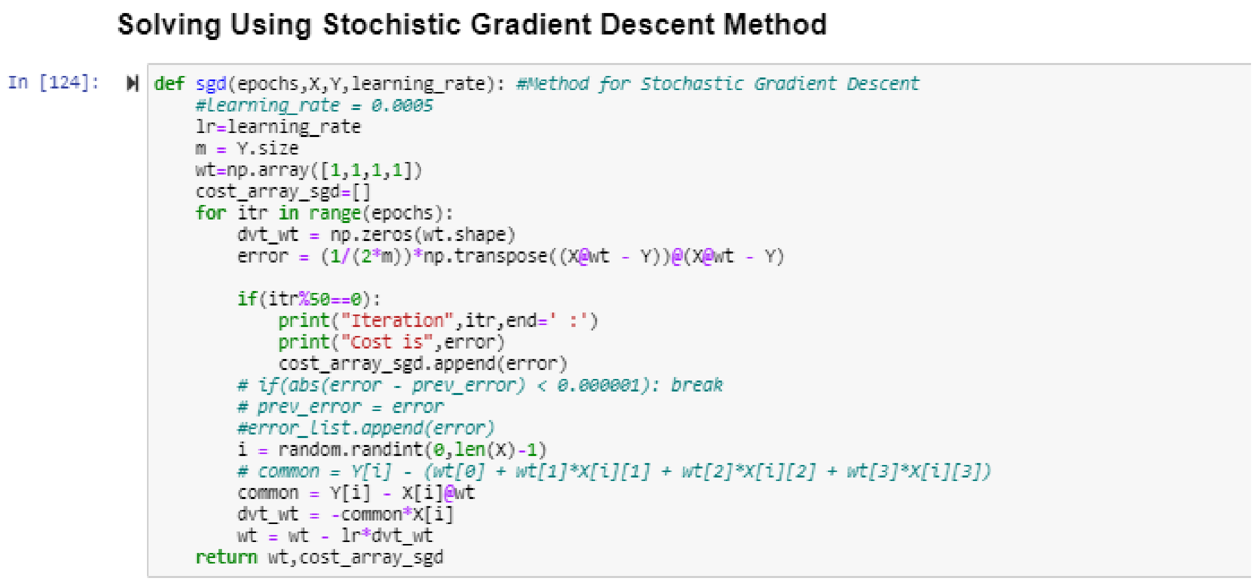
Solving using Gradient Descent Method

```
In [104]:  def gradient_descent(X, y,alpha):
               cost_array=[]
               theta=np.array([1,1,1,1])
               #alpha = 0.00001
               i=1
               max_itr=10000
               prev_cost=0
               m = y.size
               while 1:
                   cost, error = cost_function(X, y, theta)
                   if(i>=max_itr):
                       break
                   if(i%50==0):
                       print("Iteration",i,end=' :')
                       print("Cost is",cost)
                   i=i+1
                   theta = theta - (alpha * (1) * np.dot(X.T, error))
                   cost_array.append(cost)
                   if(abs(prev_cost-cost)<=1e-6):
                       break
                   prev_cost=cost
               return theta,cost_array
```

Fig. Gradient Descent Algorithm

3. **Stochastic Gradient Method:** SGD it picks up a "random" instance of training data at each step and then computes the gradient making it much faster as there is much fewer data to manipulate at a single time, unlike Batch GD.

$$\theta_j = \theta_j - \alpha\,(\widehat{y^i} - y^i)\,X_j^i$$



Solving Using Stochistic Gradient Descent Method

```
In [124]:  def sgd(epochs,X,Y,learning_rate): #Method for Stochastic Gradient Descent
               #learning_rate = 0.0005
               lr=learning_rate
               m = Y.size
               wt=np.array([1,1,1,1])
               cost_array_sgd=[]
               for itr in range(epochs):
                   dvt_wt = np.zeros(wt.shape)
                   error = (1/(2*m))*np.transpose((X@wt - Y))@(X@wt - Y)

                   if(itr%50==0):
                       print("Iteration",itr,end=' :')
                       print("Cost is",error)
                       cost_array_sgd.append(error)
                   # if(abs(error - prev_error) < 0.000001): break
                   # prev_error = error
                   #error_list.append(error)
                   i = random.randint(0,len(X)-1)
                   # common = Y[i] - (wt[0] + wt[1]*X[i][1] + wt[2]*X[i][2] + wt[3]*X[i][3])
                   common = Y[i] - X[i]@wt
                   dvt_wt = -common*X[i]
                   wt = wt - lr*dvt_wt
               return wt,cost_array_sgd
```

Fig. Stochastic Gradient Descent Algorithm

The minimum error calculated using different methods are:

1. Normal Equations Method
   Testing: 0.8685380462455952
   Training: 0.9121159876066268
2. Gradient Descent Method
   Testing : 0.8796000109152436
   Training : 0.9121715226609198
3. Stochastic Gradient Descent Method
   Testing : 0.8792900476485498
   Training: 0.9122272934810267

Visualizing the Error Vs Iterations Graph for different methods separately

Error Vs Iterations graph between Gradient Descent and Stochastic Gradient Descent was plotted and is shown below. It shows that the convergence of Stochastic Gradient descent is faster compared to Gradient Descent.

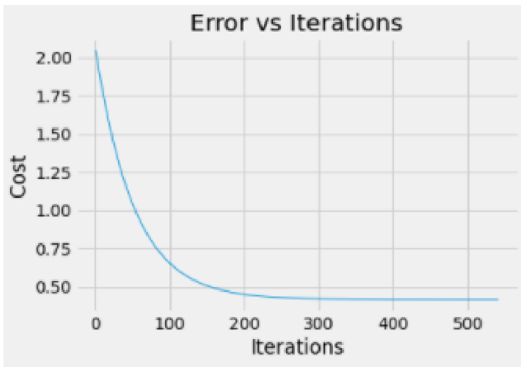


Fig. Error Vs Iterations for GD

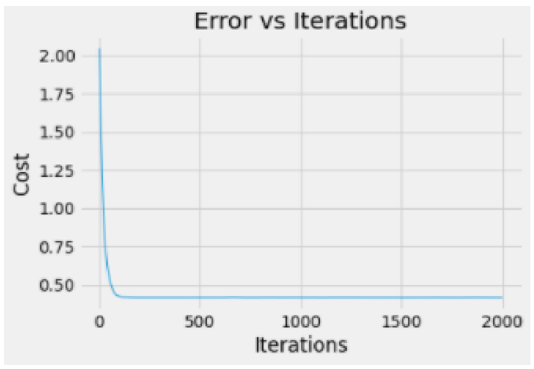


Fig. Error Vs Iterations for SGD

Error Vs Epochs Graph for different learning rates for Gradient Descent Algorithm

The choice of learning rate is very important for gradient and stochastic gradient descent method. A very high learning rate, the function may fail to converge and overshoot the minimum. If learning rate is very slow, the function takes very long time to converge and so if functionally very expensive. The Gradient Descent method was run for 4 different learning rates. The learning rates are as follows:

1. Alpha = 0.1 : This learning rate is very high and caused the function to overshoot the minimum. As a result, it gave infinitely large value in the y axis.
2. Alpha = 0.00001 : This learning rate is adequate as seen from the graph below. Our model works well for this learning rate. It is seen as the blue line in the graph below.
3. Alpha = 0.000005 : This learning is not as good as the one mentioned above as seen from the red line in the graph below.

4. Alpha = 0.000001 : This is a poor learning rate. As this learning rate is very small, the computation is very expensive and the function is taking very large time to converge. It is seen as the yellow line in the graph below.

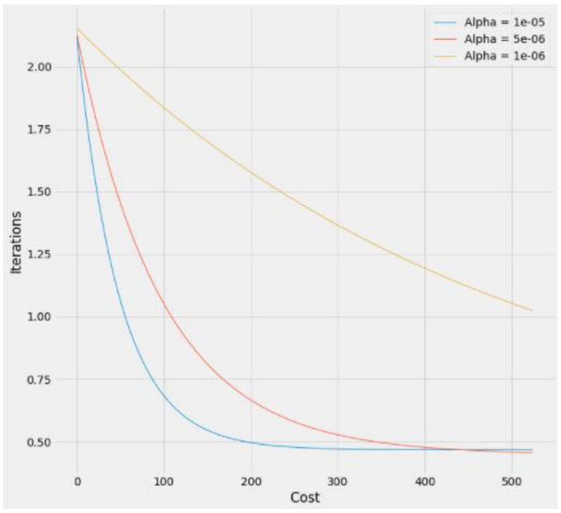


Fig. Cost Vs Iterations for GD

Error Vs Epochs Graph for different learning rates for Stochastic Gradient Descent Algorithm

3 different models were made for different learning rates. It was observed that the function still converged faster that Gradient Descent Algorithm. The 3 learning lates were: 0.001, 0.0005, 0.00001

1. Alpha = 0.001 : This learning rate is good and the function converged quickly, as seen from theblue line in the graph below.
2. Alpha = 0.0005 : The plot of this learning rate is similar to the one mentioned above. It is seen as red line in the graph below.
3. Alpha = 0.00001 : This learning rate is not good compared to the above 2 as the function is not converging properly.

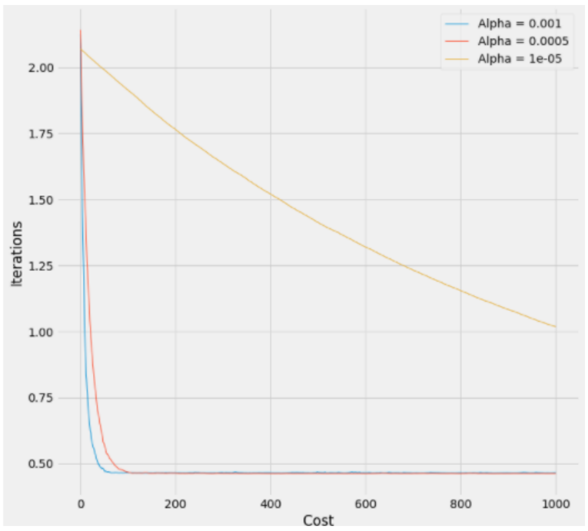


Fig. Cost Vs Epochs for SGD

Questions to Ponder On

Q1. 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?

Ans1) No, all the 3 methods do not give the same result. The most accurate result is shown by the normal equations method with least error as seen from the code. Then Gradient method calculates the weights very close to the normal equations method, however if learning rate is very less, it becomes computationally very expensive. Stochastic gradient method calculates the weights with a little more deviation from the actual weights as seen from the errors calculated. But this method is computationally better compared to Gradient Descent method.

If the size of the dataset is very large, Gradient Descent method is not recommended and it is better to use Stochastic Gradient Descent method. If the size of the dataset is very small and accurate results are expcted, normal equations method can be used. For medium sized dataset, Gradient descent works best.

Q2) How does normalization/standardization help in working with the data?

Ans 2) Data preprocessing is extremely important because it allows improving the quality of the raw experimental data. The primary aim of preprocessing is to minimise or, eventually, eliminate those small data contributions associated with the experimental error.

Q3) 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})

Ans3) As the number of training iterations increase, the cost value decreases. But for very large values of epochs (~10^{10}), the loss saturates and the line almost becomes parallel to the x-axis.

Q4) What primary difference did you find between the plots of Gradient Descent and Stochastic Gradient Descent?

It was observed that for same learning rates for both gradient descent and stochastic gradient descent, the gradient descent converged more compared to stochastic gradient method. For higher learning rates, stochastic gradient converged faster and better.

Q5) What would have happened if a very large value (2 or higher) were to be used for learning rate in GD/SGD?

Ans) Very Large values of the learning rate in both GD and SGD can cuase the function to overshoot the minimum and the function may fail to converge. A very high learning rate causes the Cost to become infinite and so, the graph shows infinite value in the y-axis.

Q6) 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 $Y = W^T X$ instead of $Y = W^T X$ + B

Ans) The minimum error will change if the bias term was removed from the equation. Without the bias term, the effect of the bias term would add with the total error of the model and so the minimum error obtained will be more compared to the model where bias was used.

Q7) 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 weights vector tells us the influence of a particular factor (independent variable) on the result (dependent variable). For example, taking the linear relation to be y=0.25 + 5.5x1 + 0.01x2. A high value of 5.5 associated with independent variable x1 tells that this term contributes more to the final valye y, compared to the weight of 0.01 of x2.

In our model, it was observed that Age was the most important variable, and then bmi in determining the charges.

Minimum influence is caused by number of children.