UNIVERSITY ADMISSION PREDICTION USING LINEAR REGRESSION

Submitted by Venkata Lahari Balantrapu

Linear regression is a regression model that assumes the relationship between the dependent variable y and an independent variable or the input variables x. The relationship between the y and x can be expressed as shown in the figure below where E is the noise.

$$\hat{y}_n = \theta_0 + \sum_{i=1}^{M} \theta_i x_{ni} + \varepsilon_n,$$

In this report the linear regression has been applied on the university admission data set along with the three gradient descents algorithms i.e., gradient descent, stochastic gradient descent, mini-batch gradient descent.

Gradient Descent:

The gradient descent is an optimization algorithm that is used to find the coefficients that would help minimize the cost of the function and the idea behind this is to loop over different coefficients evaluating their costs and selecting the coefficients that have the lower cost. This repeated process over a certain number of times will lead to a local-minima i.e., values of coefficients that would result to a minimum cost.

Three types of gradient descents will be discussed in this report, they are:

1. Standard Gradient Descent
2. Stochastic Gradient Descent
3. Mini-Batch Gradient Descent

In the standard gradient descent, the error has been calculated for each and every training example and the model gets updated only after all the training samples have been evaluated. The downside to this model is that with a data set of higher dimensionalities the computational time and memory intensifies. The entire training data set should be available in the memory for this algorithm. Also, this model at times gets into the convergence of lower accuracy.

Stochastic gradient descent and the mini-batch gradient descent are used in the place of gradient descent to overcome its drawbacks. In the stochastic gradient descent only one sample at random will be passed through the network and its cost will be computed. Only one training sample would be loaded into the memory at a time. Although this makes stochastic gradient descent fast, the repeated updates may lead to some noise in the gradients which might affect the error have abnormal jumps. Furthermore, as it deals with single sample vectorized operations cannot be performed.

Stochastic gradient descent can be considered as mini-batch gradient descent of batch size equal to 1. In mini-batch gradient descent we divided the data set into several mini batches. We calculate the cost over the loop of every single batch. This helps in reducing the noisy gradients also avoiding the local minima.

Methodology:

The data set has been split into training and testing data sets of 75:25 where 300 samples are used to train the model and 100 samples to test the model.

The gradient descent is applied to optimize the sum squared error (SSE), where sum squared error is the summation of the squares of the differences between the predicted and the actual value. The sum squared error is expressed as shown in the equation below,

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2 = \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{x}_n^{\mathsf{T}}\boldsymbol{\theta} - y_n)^2$$

This cost is optimised by using the three different gradient functions, where in the standard gradient function the cost is calculated for the whole training samples over a number of epochs. Where as in the mini-batch gradient descent the cost will be divided by the batch size.

Root mean Squared and R square: These are the two metrics used to evaluate the performance of the test data set of 100 samples.

Root mean squared is a measure of fit which indicates the distance of the observed data points to the predicted values. Lower values of RMSE indicates better performance and fit of the model whereas R square is a measure that indicates how close the data points are fitted to the regression line. The higher the values of R square the better the model is fitted. RMSE is expressed as the square root of the variance of the residuals.

$$\mathbf{RMSD} = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}}$$

$\mathbf{RMSD}$ = root-mean-square deviation
i = variable i
N = number of non-missing data points
x_i = actual observations time series
$\hat{x}_i$ = estimated time series

Formula ›

$$R^2 = 1 - \frac{RSS}{TSS}$$

Figure1: The formulae for RMSE and R square

In the above figure, RSS is the sum squared error and TSS is the total sum of squares.

Hyper Parameters: Hyper parameters play a very important role in the optimisation using the gradient descent. The major hyper parameters that will be discussed in this report are learning rates, the epochs and the batch size.

All the hyper parameters have been manually tested to find the optimal one's that would lead to the best performance.

Epochs: Epochs are the number of times a model is looped over the training data the higher values of epochs generally lead to the over fitting of the data and the lower values of it would make the data underfit. Choosing the right amount of the epochs is necessary to achieve better performance. Each and every gradient descent is looped through the set of epochs ranging from 40 to 100. The learning rate is kept constant at 0.001 and the batch size for the mini-batch gradient descent is set to 32.

From the below figures it can be observed that there is not much difference in the performance of the stochastic gradient descent and the standard gradient descent but where as for the mini-batch the

r square gradually increased over the number of iterations before it started going down. The epoch value of 90 is chosen as a middle ground also it is observed that epochs over 100 made the gradient descent over fit the data.

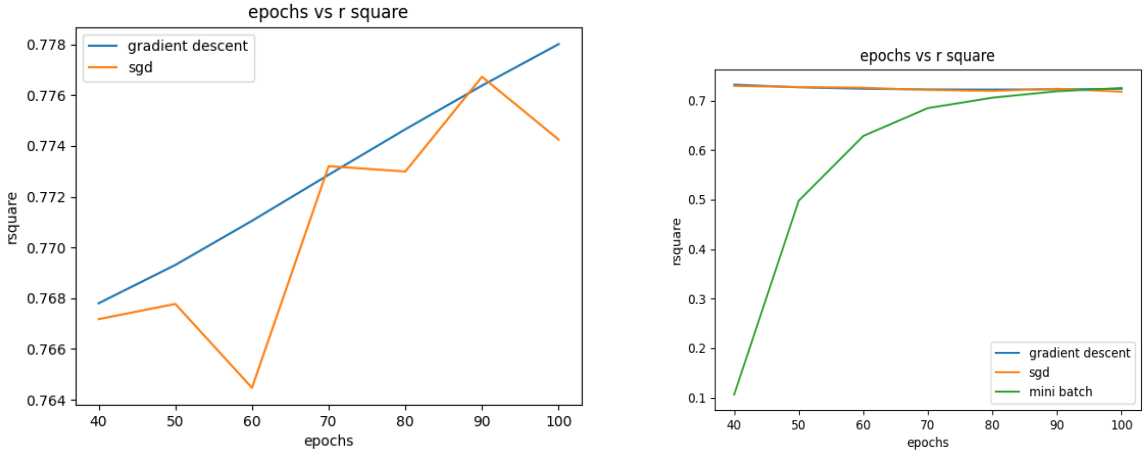

Figure2: Epoch vs r square

Learning Rate: Learning rate is tuning parameters that defines the speed at which the model learns. The learning rate plays a really major role in determining the performance of the system. In this report the model is looped through a set of learning rates ranging between 0.001 to 0.08.

For the learning rate the standard gradient descent overfit the data and the performance degraded over any learning rate above 0.001 as shown in the figure below.

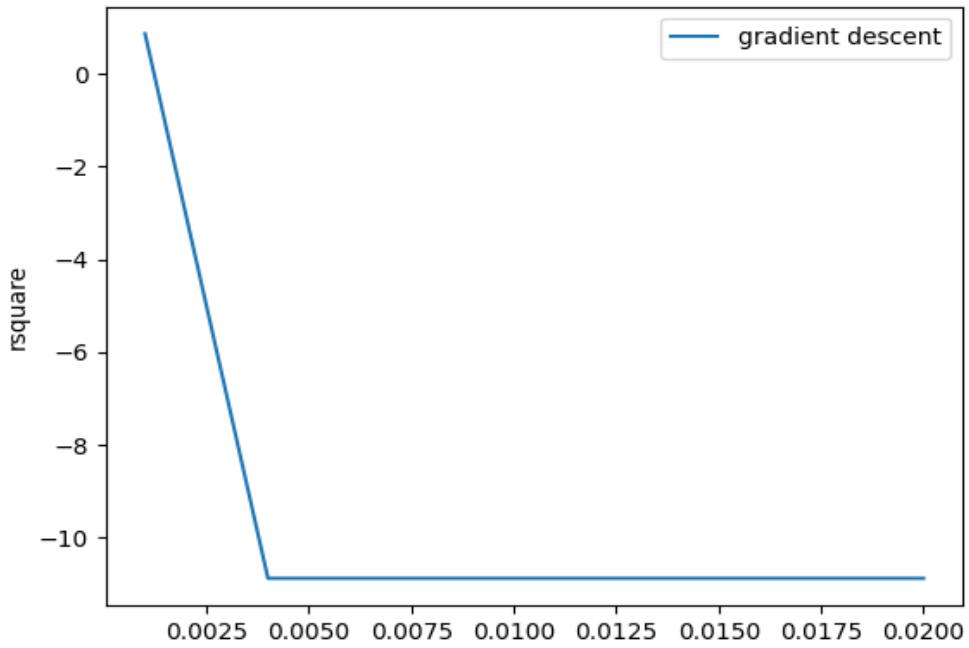

Figure 3: Learning rate vs r square for the standard gradient descent

As for the stochastic gradient descent the highest r square was observed at the learning rate of 0.01 and the mini batch had an improvement in the performance at higher learning rates. During this

process the epochs have been set to 90 and the batch size for mini batch gradient descent has been set to 32. The mini batch had the best performance at the learning rate of 0.07.

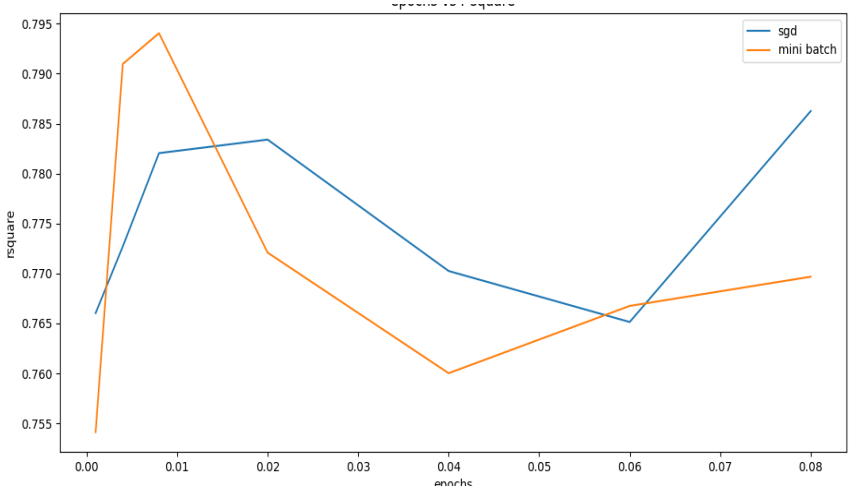

Figure 4: Learning rate vs r square for stochastic gradient descent and mini batch

Batch Size: Batch size is the number of mini batches to be made for the data to iterate and the cost to be evaluated. All the models have been tested over the batch sizes of 20, 32 ,50, 64.

The mini batch gradient descent has been looped through different batch sizes over a constant learning rate and iterations or epochs to check the performance. From the figure it can be observed that the model started to over fit after a batch size of 15 and had a performance decay. This could be expected cause the data set in general has less samples. Furthermore, after subjecting the model to a batch size of t 10 and 15 the model had a seemingly better R square and minimal RMSE at the batch size of 15 which has been chosen in the final model.

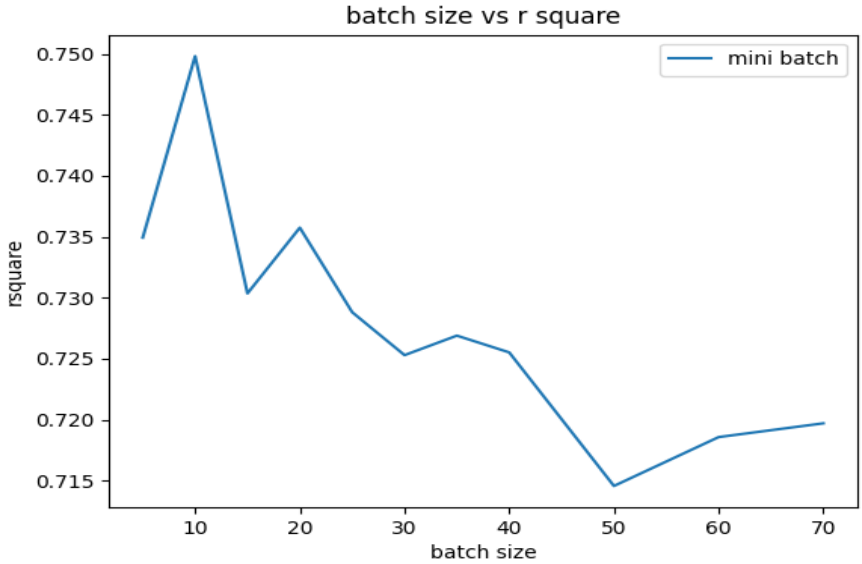

Figure 5: Batch size vs R square for Mini-batch Gradient descent

After experimenting with all the hyper parameters, the optimal parameters have been chosen and applied on the final model to give the best performance with an optimised cost using the gradient

descents along with a higher R square. The final performance of the three gradient descents on the model can be seen in the picture 6.

```
 RESTART: C:\Users\lahar\AppData\Local\Programs\Python\Python37-32\linear_reg.py

Warning (from warnings module):
  File "C:\Users\lahar\AppData\Local\Programs\Python\Python37-32\linear_reg.py", line 84
    x['intercept']=1
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
Gradient Descent
at 90 epochs and 0.001 alpha the cost is 0.18279402718919027
at 90 epochs and 0.001 alpha the RMSE is 0.0060463878008144714
at 90 epochs and 0.001 alpha the R suqare is 0.7957735050319841
Stochastic Gradient Descent
at 90 epochs and 0.01 alpha the cost is 0.1651868161040897
at 90 epochs and 0.01 alpha the RMSE is 0.005747813777499924
at 90 epochs and 0.01 alpha the R square is 0.8232827268479987
Mini-batch Gradient Descent
at 90 epochs,batch size of 15 and 0.07 alpha the cost is 0.16645752023235272
at 90 epochs, batch size of 15 and 0.07 alpha the RMSE is 0.005769879032221607
at 90 epochs, batch size of 15 and 0.07 alpha the R square is 0.8207244639941725
>>>
```

Figure 6: Final results after optimization using three gradient descents.