MACHINE LEARNING ASSIGNMENT 1 REPORT

In this assignment, in the first question, we were asked to implement linear regression from scratch using only numpy and pandas libraries in python. And in the second question we were asked to do the same, but using some pre-built(library based) implementation of linear regression and do a comparative analysis between the performance of both the models on the test dataset and also on some individual examples.

DATASET DESCRIPTION:

The dataset consists of 10000 rows and 6 columns. Among which **5 columns** are considered as **independent variables (Hours of study, Previous scores, Extracurricular activities, Duration of sleep and sample question papers practised).** The 6th column i.e. Performance of the student is the **dependent variable** as we are studying the effect of the other parameters on this specific parameter.

While analysing the data, we faced two problems:

1. The **“Extracurricular Activities”** column contained values in **“Yes”** and **“No”** only, which was not a supported format if we want to implement matrix operations on the dataset.
   Solution: We changed the values to **1 for “Yes”** and **0 for “No”** using the `.replace()` method in pandas.

2. The “**Previous Scores**” column had values which were way larger than the values of other columns. This was creating problems while training the model as the model was not performing well even for **learning rate as small as 0.001** and **20,000 epochs.**
   Solution: We standardised the values of that particular column such that the mean is 0 and the standard deviation is 1.

Question 1:

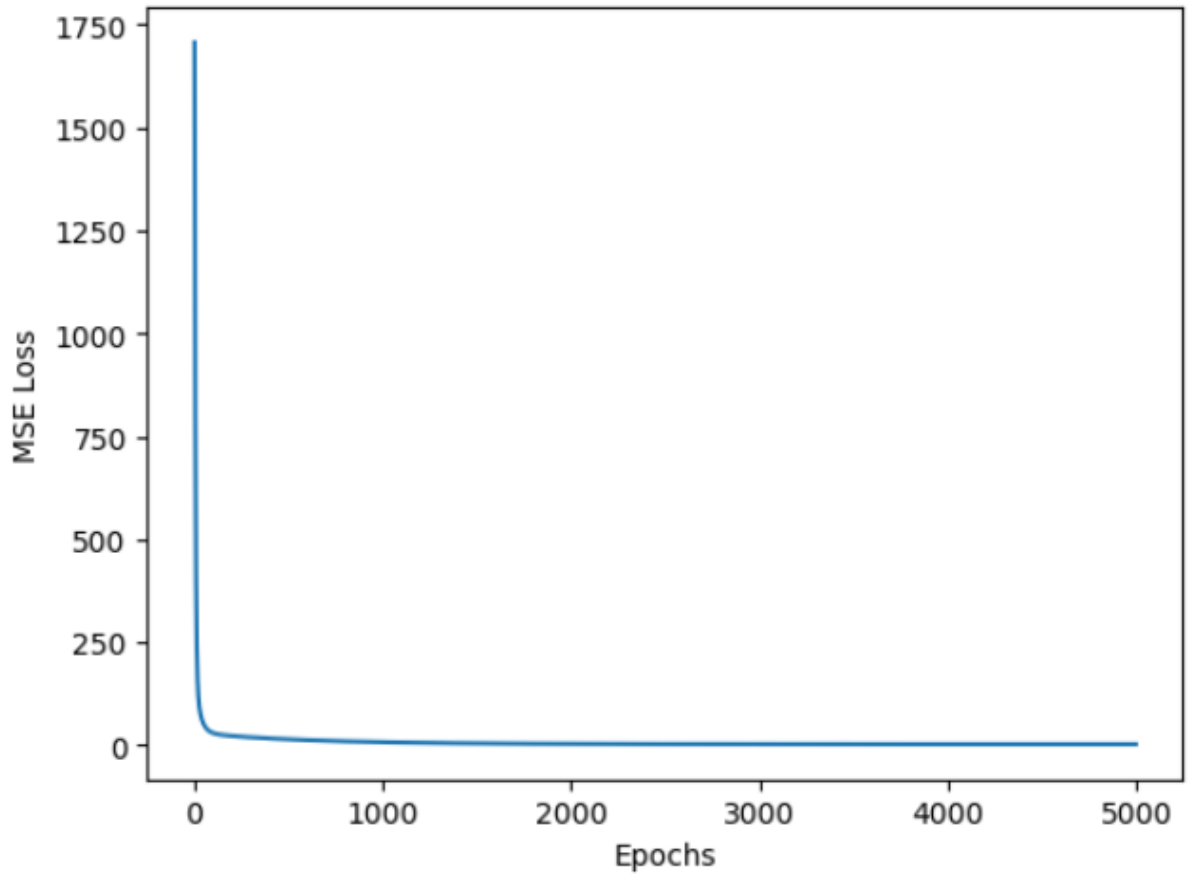
Colab link:
https://colab.research.google.com/drive/1bShSKPkimnxQBhO8gZcT17bIsGzfLW0C?usp=sharing

In this question we had to implement linear regression from scratch using only the numpy and matplotlib library. So after the data pre-processing, We implemented the **`linearRegression()`** function which took **`X_train`** and **`y_train`** as input (i.e. the training data). The function took the number of epochs and the learning rate as input from the user and then returned the optimised weights and bias to the user.

For optimising the weights and bias we have used gradient descent which gets updated using the whole dataset(batch gradient descent) in every epoch.

The final values of weights and bias after 5000 epochs with a learning rate of 0.02 were as follows:

Weights : `[ 2.86702974 17.65447502 0.68318252 0.52318604 0.20289693]`
Bias: `36.19034308423`
The MSE loss vs epoch graph looks like the following:



The `predict()` function, given the test data, weights and bias returns the value of `y_predicted`, the predicted output of the model.
For evaluating the performance of the model, we have also implemented the mean square error function(`evaluate_mse`) and R2 score(`evaluate_r2_score`)

Question 2:

Colab link:
https://colab.research.google.com/drive/1wSxYCjYK8Zay2FznjPcn8cMBKv6gbVdu?usp=sharing

This question required us to use a pre-built implementation of linear regression in python, so I have chosen the `LinarRegression` model of Scikit-learn, a python library for implementing machine learning algorithms. For evaluation of the performance of the algorithm we have taken `mean_squared_error` and `r2_score` from `sklearn.metrics`.

Comparative analysis of the two implementations:

In the below table we have shown the comparative studies of the mean squared errors and the R2 scores for both the implementation from scratch and the scikit learn based implementations.

Performance metric	Implementation from scratch	Scikit learn based implementation
Mean squared error	4.2500414	4.2471829
R2 score	0.9883620	0.9836991

As we can see both the models performed almost similarly for this dataset. But as the dataset was more or less an ideal dataset with no abnormalities, this result was expected. We can expect better results from the scikit learn based implementation in case of more skewed datasets.

Areas of improvement:

We can improve the working of the algorithm we focus on these following things:

1. Experimenting more with the learning rate and epochs to fine tune the value of the weights and bias.
2. Adding a cross validation set to train the model on different subset of the same data.
3. For linear regression we can use the Normal formula instead of gradient descent to find the most optimized value of weights and bias.
4. Although marginal for this particular dataset, feature scaling in general can improve the performance of the algorithm.