ECON 425 – Homework 2 Write-Up

By: Anshika Sharma, UCLA ID(305488635)

Part -1

Background

For this part we look at the math and verbal SAT scores and GPA scores of 105 students. We will fit a supervised learning model (in the form of a linear regression) in order to predict student GPA scores given the SAT scores.

Algorithm

First, I imported all the packages and in-built functions.

For this part, I loaded the data that contains GPA and SAT scores. Then, I only keep the data that is of interest (i.e. the math SAT, verbal SAT, and GPA scores).

Then, I normalize the data using definition 'meanNormalization'. I store the new values into a new 'sat1' matrix. I normalize the data using 'rescaleNormalization' and store normalized values in 'sat2' matrix. I normalize it using 'rescaleMatrix' and store normalized values into 'sat3' matrix **(carried out in placeholder-2)**. For Rescale normalization and mean normalization, since they take in 1-d array, we need to create a loop to apply the function to each column. Normalization of the data is necessary as we want our data to satisfy standard linear regression assumptions. The dataNormalization.py file contains three functions to perform Data normalization i.e 'rescaleNormalization' , 'rescaleMatrix' and 'meanNormalization' all of which were tried by me turn by turn on the entire system.

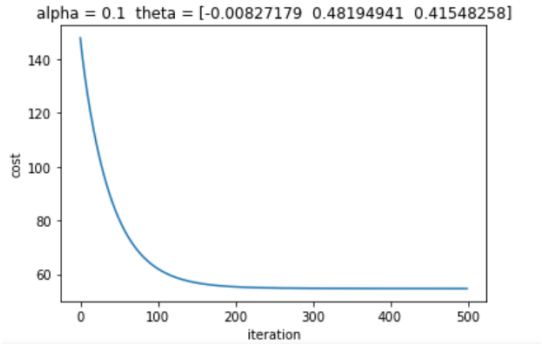
Following this, I split the data into training (60 observations) and testing (45 observations) data and define parameters. Note that since I use all the three normalization functions turn by turn, I create three training and three testing data sets accordingly. I call the gradient function **(In place holder 4 and 5)** which will update estimated parameters in order to find the optimal values, and then compute the cost function. The cost function is then plotted to determine where it converges (I.e. the minimum value). You can change 'ALPHA' and 'MAX_ITER' **(in placeholder 1)** to get different curves. I did this for all the three normalization techniques. Finally, I test the fitted regression, given by the optimal parameters found from the gradient descent definition, on the testing data and computed the average error and standard deviations. I also computed the R^2 by hand using a for-loop. **(carried out in placeholder – 3)**

The algorithm produces the following results for different iterations and alphas and different normalization techniques.

Graph for Mean Normalization technique:

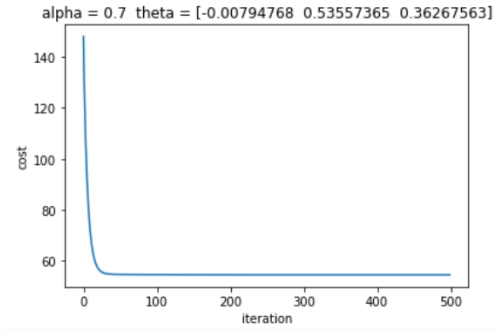
Changing alpha and fixing iterations

<u>For alpha = 0.1 and max_iter = 500</u>



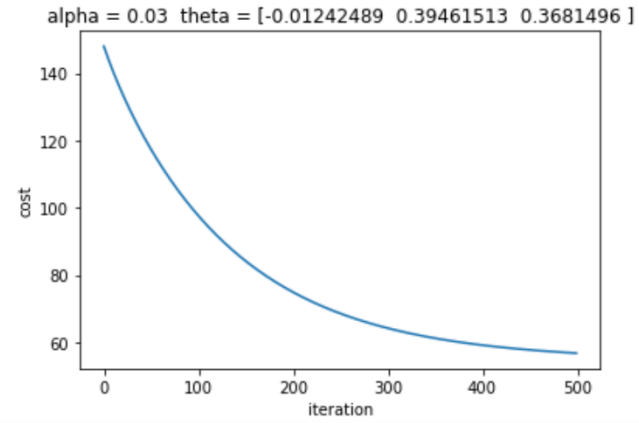
results: 0.17326009831040687 (0.12729258175503566)
R2 is -0.07487817407505704

For alpha = 0.7 and max_iter = 500



results: 0.1746711527870884 (0.1272265387827901)
R2 is -0.08590405868564144

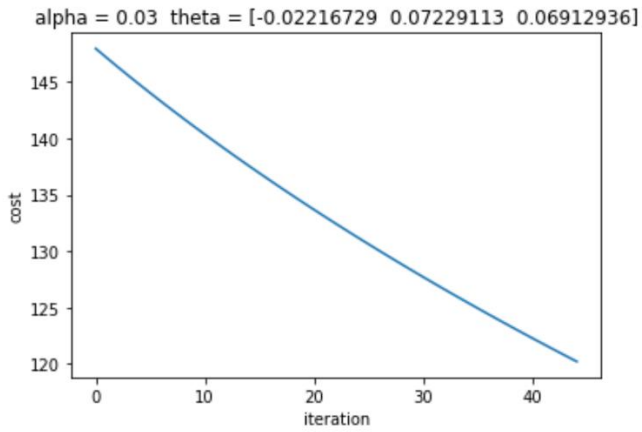
For alpha = 0.03 and max_iter = 500



results: 0.16007231938551486 (0.12158257572740852)
R2 is 0.06039282355969666

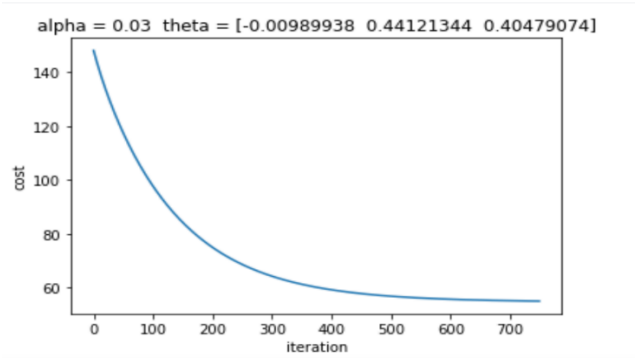
Changing the number of iterations and fixing alphas:

For alpha = 0.03 and max_iter = 45




```
results: 0.1576453020632505 (0.1273488229842768)
R2 is 0.044945032900803095
```

<u>For alpha = 0.03 and max_iter = 750</u>

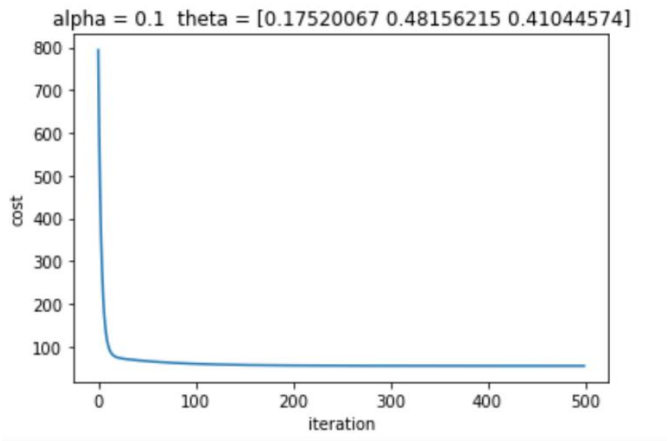



```
results: 0.16795316958181583 (0.12444542689420444)
R2 is -0.016101805167742178
```

Graph for Rescale Normalization technique:

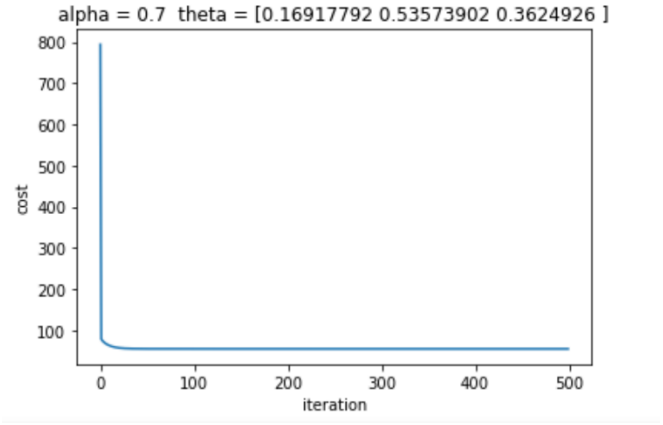
Changing alpha and fixing iterations:

<u>For alpha = 0.1 and max_iter = 500</u>



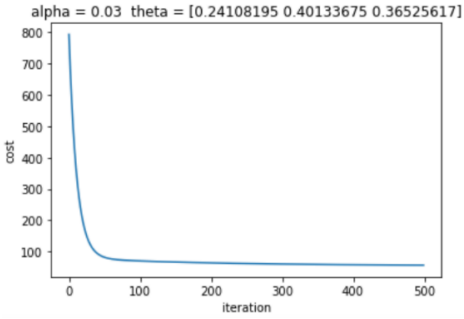
```
results: 0.1727749787556687 (0.12688442297897462)
R2 is -0.06856196284761995
```

For alpha = 0.7 and max_iter = 500




```
results: 0.17467368070926126 (0.1272258039446234)
R2 is -0.0859202469314726
```

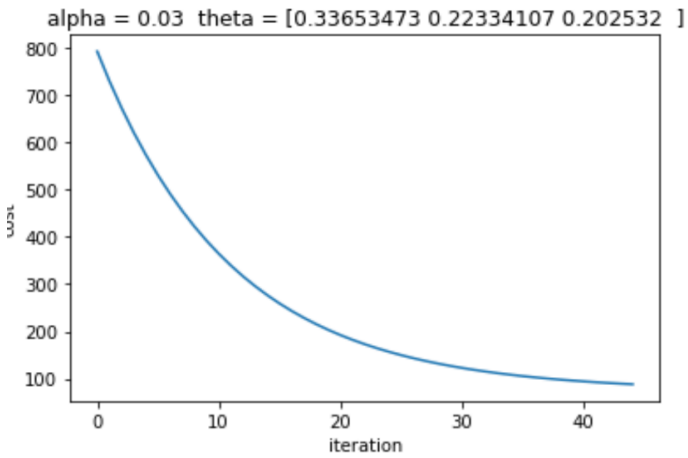
For alpha = 0.03 and max_iter = 500




```
results: 0.1600384580086267 (0.12094515356900636)
R2 is 0.06423985312728175
```

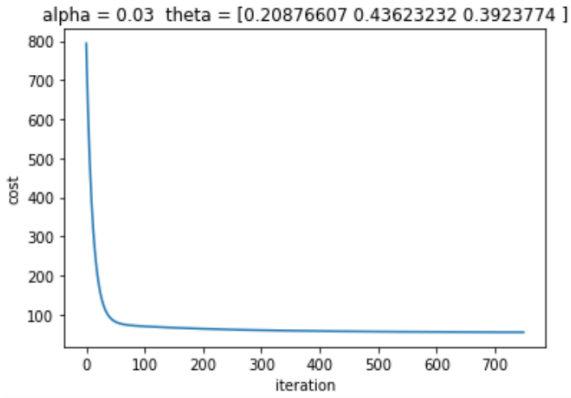
Changing the number of iterations and fixing alphas:

For alpha = 0.03 and max_iter = 45




```
results: 0.17710891801697098 (0.11686115812343202)
R2 is -0.04701087461303621
```

For alpha = 0.03 and max_iter = 750




```
results: 0.16615439786671954 (0.123299393199903)
R2 is 0.004476228172385577
```

Note: The graph for Rescale Normalization technique as well as the results are exactly the same as that of Rescale Matrix because their way of calculation for normalization is exactly the same.

Given the results, there are a few things to note. First, the change in alpha will result in a change in the curvature for the convergence graph. Specifically, if the alpha is higher there will be a steeper curve while a smaller alpha will result in a slower or more gradual convergence. Changing the number of iterations will simply extend the curve out given an increase in iterations, or shorten a curve with a decrease in iterations. Changing either the number of iterations or the alpha doesn't significantly impact the average error or standard deviation, however there is a slight and not very significant difference in the R^2. Notice that the curves tend to asymptote to zero. Note that when the R^2 is negative for the testing sample it suggests that it performs even worse than the mean and that the model does not do a very good job in predicting the data.

The dataNormalization.py file contains three functions to perform Data normalization i.e 'rescaleNormalization' , 'rescaleMatrix' and 'meanNormalization'. There is a difference of python data

structure between rescaleNormalization and rescaleMatrix. The former takes in a data as a 1-d array whereas the latter takes in a matrix (2-d array). The operation performed by both are essentially the same. It calculates the max and min values of each type of score. Then, from each value, it subtracts the minimum value and divides by the range in order to normalize the data. Comparing 'rescaleMatrix' and 'meanNormalization', apart from the fact that the former takes in data matrix and latter takes in data array, the operation performed by the functions is different. The 'meanNormalization' computes the mean, minimum, and maximum of each type of score. Then for each observation, it subtracts the mean and divides by the range in order to normalize the data.

The system performs differently over testing samples while using different normalization functions. Comparing 'meanNormalization' and 'rescaleNormalization', for the same values of alpha and max_iterations, the convergence curve suggests that the convergence for rescale normalization is faster and it is steeper. The same result was delivered taking different set of values of alpha and max_iterations. Comparing 'meanNormalization' and 'rescaleMatrix', for the same values of alpha and max_iterations, the convergence curve for the latter is steeper and converges faster as well. The same result was delivered taking different set of values of alpha and max_iterations. Comparing 'RescaleMatrix' and 'RescaleNormalization', there isn't any difference in the graphs as expected since the only difference b/w the methods is the data type as I mentioned above. Comparing the impact of different normalization techniques on average error and standard deviations, it doesn't impact it significantly. The R2 isn't also impacted except the case where we increase the iterations. In that case, the R2 for Rescale normalization is a very small positive value whereas that for Mean normalization is a very small negative value, only difference being that of the sign. The R^2 for Mean normalization is highest when alpha = 0.03 and lowest when alpha = 0.7. For Rescale Normalization technique, the R^2 is highest when alpha is 0.7 and lowest when 0.03

Part – 2

Background

There exists a problem of overfitting in this dataset as the relationship of testing dataset has weaker relationship with dependent variables. Hence, in part 2, the regularization will be applied to overcome this problem. Ridge, Lasso, and Elastic-Net models are forms of regularized linear techniques found in General Linear Models. They add different penalties to the regression function. Different values of hyperparameters are used to find the best model for each technique, as supported by the table in the appendix.

Algorithm

For this part, I create a separate ipynb file named "main_ha2_partII". So, in the first step, I import all the packages and then load in the sat data set. As before, I only keep the data that is of interest (i.e. the math SAT, verbal SAT, and GPA scores).

Then, I normalize the data using minmax_scaler from sklearn.preprocessing package. I split the data into training and testing data set in the same way as I did in part-1 as required by the question.

First, I carried out Linear Regression using the sklearn.linear.model package. Then, I find the training and testing score and intercept and coefficients of the model and convert it into a data frame.

Next, I implement Ridge regression, Lasso regression and Elastic net regression using the Scikit Learn. For each, I generate a function that intakes the training and testing data, alpha values and maximum

iterations. It then runs the respective regression, calculates the training and testing score and intercept and coefficients and appends it in a list and returns a list.

Then, with the help of the function, I generate a matrix containing training, testing score and all the coefficients for different values of alpha and other hyper parameters as well like lr_ratio (eg, in case of lasso regression and linear regression using gradient descent). After that, I extract the row from the data frame that gives the most optimal R^2 and name it the "best model" accordingly. I repeat the same for each model and also for the Linear regression model from part-1 that was developed from scratch using gradient descent. (named "lr_scratch", it has hyper-parameters that are used as arguments in gradient descent function)(Other names are self explanatory).

The following table does the comparison b/w the optimal testing scores or R^2 for all the best models and Linear regression using scikit learn. It also compares training score, intercept, coef_x_1 and coef_x_2.:

	Model	training_score	testing_score	intercept	coef_x_1	coef_x_2
formular	linear regession	0.6269	-0.08691	0.169	0.5404	0.3578
alpha_5	Ridge_model	0.5259	0.1717	0.343	0.2802	0.2602
alpha_0.01	Lasso_model	0.6072	0.06928	0.2441	0.4695	0.2691
alpha_0.1	Elastic_model	0.4807	0.1782	0.3781	0.2424	0.224
learning_rate_0.001	lr_scratch	0.5986	0.1062	0.268	0.3735	0.3413

Note: This table contains values of "the best models" or, in other words, which have the most optimal R^2, corresponding to different values of hyper parameters and linear regression model using scikit as well.

We begin with Linear regression using sklearn package and obtain an optimal R square of -0.08. There wasn't much to explore with this model and it cannot be influenced much since we are using the package in this case.

Ridge regression minimizes the size of all coefficients by adding a factor of sum of squares of coefficients in the optimization objective. When alpha is small (nearly equal to zero), the model returns the coefficient and accuracy score as generated by the the simple linear regression (the one without any penalty), as seen in the table. When alpha increases, the coefficients become smaller; coef_1 and coef_x_2 = (0.54, 0.357) and (0.28, 0.26), when alpha is equal to 0.0001 and 5, respectively. The accuracy score on testing also improves suggesting that overfitting problem is solved. However, the accuracy score shows the downward trend after alpha equals to 5. A large value of alpha can lead to underfitting of the model and making the parameters converge to zero.

Lasso stands for Least Absolute Shrinkage Selector Operator. Lasso assigns a penalty to the coefficients in the linear model by adding a factor of sum of absolute value of coefficients in the optimization objective and eliminates variables with coefficients that are zero. This is called shrinkage or the process where data values are shrunk to a central point such as a mean. Lasso adds a penalty to coefficients the model overemphasizes. This reduces the degree of overfitting that occurs within the model. The effect of changing alpha is quite similar to ridge regression; it returns the same result as simple linear regression when alpha is set to zero and shows a better accuracy score on testing set when alpha increase until a certain value. However, given same value of alpha, lasso returns a smaller value of coefficient compared to the ridge model. Moreover, the coefficients can become zero when alpha is set to 1, unlike ridge model which does not reach to zero although alpha converges to infinity.

Thus, Lasso model can use feature selection which is not the main aim of this case because we have only two features. Lasso does not work well with multicollinearity. Lasso might randomly choose one of the multicollinear variables without understanding the context. Such an action might eliminate relevant independent variables.

Elastic-Net regression combines the L1 and L2 penalties of both ridge and lasso methods. In this model, apart from alpha, there is another parameter, l1_ratio, weight assigned to lasso penalty. The effect of changing alpha is similar to other methods. Given alpha, the value of coefficient is between the value created by ridge and lasso model. Also, the coefficient is zero when alpha = 1, which larger than that of the lasso model where alpha equal to 0.1. Taking 0 < l1_ratio < 1, the model gives a higher accuracy score when l1_ratio is smaller (weighing more on a ridge model). The "best model"(the model having optimal R^2; concept of best model explained in algorithm) created by the elastic-net model is when the l1_ratio = 0.1 and it gives a little improvement of accuracy score (0.17182) when compared to the best model from the ridge model (0.17166).

In the linear regression using gradient descent method, named lr_scratch Model, the parameter in this model is learning rate. Taking iteration as equal to 5000, for a very small value of learning rate, we get a negative value of accuracy score for the testing set, implying that there is a very slow convergence rate of theta and more iterations are needed to give a better accuracy. But, when 0.01 < Learning rate < 1, the accuracy score of testing set has smaller negative value, although the score of training set is high.

Comparing the simple linear regression model and the best models, that is the models with the highest testing score formed using the other models, the simple linear regression has the worst performance due to overfitting. The best Lasso regression model has a low accuracy score of 6.92%, inspite of applying a penalty function. Unpredictably, the linear regression using gradient descent method gives a positive accuracy score although it does not penalize the impact of coefficient. The elastic net model gives the best performance followed by ridge model. Elastic net model's accuracy score is quite low , about 17.88%. The model specification should be corrected with non-linear or log function, apart from applying the regularization in order to improve the accuracy score as per me. Also, the size of the data should increase to give a better picture of the sample. If that isn't feasible, splitting data for training and testing should be randomly selected because the latter part of data set exposes a weaker relationship between dependent variables and features. K-Fold Cross Validation may be applied since it decreases the problem of bias. It also ensures that the model is robust.

TABLES:

1. **Linear Regression using Scikit learn**

	Model	training_score	testing_score	intercept	coef_x_1	coef_x_2
0	linear regession	0.626851	-0.0869148	0.168953	0.540353	0.357821

2. Ridge Model using Scikit Learn

	training_score	testing_score	intercept	coef_x_1	coef_x_2
alpha_1e-10	0.62685	-0.086915	0.16895	0.54035	0.35782
alpha_1e-08	0.62685	-0.086915	0.16895	0.54035	0.35782
alpha_1e-05	0.62685	-0.086913	0.16895	0.54035	0.35782
alpha_0.0001	0.62685	-0.086897	0.16896	0.54033	0.35783
alpha_0.001	0.62685	-0.086733	0.16902	0.54012	0.35794
alpha_0.01	0.62685	-0.085107	0.1696	0.53803	0.35898
alpha_0.1	0.6267	-0.069921	0.17523	0.5198	0.36681
alpha_1	0.61755	0.033172	0.22172	0.42899	0.36461
alpha_5	0.52594	0.17166	0.34297	0.28021	0.26017
alpha_10	0.42188	0.16794	0.41628	0.19878	0.18747
alpha_15	0.34815	0.13744	0.45701	0.15424	0.14628

	Model	training_score	testing_score	intercept	coef_x_1	coef_x_2
alpha_5	Ridge_model	0.52594	0.17166	0.34297	0.28021	0.26017

3. Lasso Model Using Scikit learn

	traning_score	testing_score	intercept	coef_x_1	coef_x_2
alpha_1e-10	0.6269	-0.08691	0.169	0.5404	0.3578
alpha_1e-08	0.6269	-0.08692	0.169	0.5404	0.3578
alpha_1e-05	0.6269	-0.08678	0.169	0.5404	0.3576
alpha_0.0001	0.6268	-0.08506	0.1697	0.5398	0.3568
alpha_0.001	0.6267	-0.06824	0.1765	0.5334	0.3488
alpha_0.01	0.6072	0.06928	0.2441	0.4695	0.2691
alpha_0.1	0	-0.1296	0.5997	0	0
alpha_1	0	-0.1296	0.5997	0	0
alpha_5	0	-0.1296	0.5997	0	0
alpha_10	0	-0.1296	0.5997	0	0
alpha_15	0	-0.1296	0.5997	0	0

	Model	traning_score	testing_score	intercept	coef_x_1	coef_x_2
alpha_0.01	Lasso_model	0.6072	0.06928	0.2441	0.4695	0.2691

4. Elastic Net Model using Scikit Learn

L1_ratio = 0.5

	training_score	testing_score	intercept	coef_x_1	coef_x_2
alpha_1e-10	0.6269	-0.08691	0.169	0.5404	0.3578
alpha_1e-08	0.6269	-0.08691	0.169	0.5404	0.3578
alpha_1e-05	0.6269	-0.08682	0.169	0.5404	0.3577
alpha_0.0001	0.6269	-0.08547	0.1695	0.5394	0.3576
alpha_0.001	0.6267	-0.0723	0.1746	0.5302	0.3566
alpha_0.01	0.6171	0.03497	0.2231	0.4545	0.333
alpha_0.1	0.09832	-0.03784	0.5641	0.0559	0.01707
alpha_1	0	-0.1296	0.5997	0	0
alpha_5	0	-0.1296	0.5997	0	0
alpha_10	0	-0.1296	0.5997	0	0
alpha_15	0	-0.1296	0.5997	0	0

	Model	training_score	testing_score	intercept	coef_x_1	coef_x_2
alpha_0.01	Elastic_model	0.6171	0.03497	0.2231	0.4545	0.333

L1_ratio = 0.3

	training_score	testing_score	intercept	coef_x_1	coef_x_2
alpha_1e-10	0.62685	-0.086915	0.16895	0.54035	0.35782
alpha_1e-08	0.62685	-0.086915	0.16895	0.54035	0.35782
alpha_1e-05	0.62685	-0.086832	0.16899	0.54038	0.35772
alpha_0.0001	0.62685	-0.08563	0.16944	0.53929	0.35793
alpha_0.001	0.62677	-0.073959	0.17387	0.52904	0.3595
alpha_0.01	0.61989	0.019335	0.21478	0.45523	0.35072
alpha_0.1	0.31984	0.12317	0.47088	0.14842	0.12181
alpha_1	0	-0.12963	0.59971	0	0
alpha_5	0	-0.12963	0.59971	0	0
alpha_10	0	-0.12963	0.59971	0	0
alpha_15	0	-0.12963	0.59971	0	0

	Model	training_score	testing_score	intercept	coef_x_1	coef_x_2
alpha_0.1	Elastic_model	0.31984	0.12317	0.47088	0.14842	0.12181

L1_ratio = 0.1

	training_score	testing_score	intercept	coef_x_1	coef_x_2
alpha_1e-10	0.62685	-0.086915	0.16895	0.54035	0.35782
alpha_1e-08	0.62685	-0.086915	0.16895	0.54035	0.35782
alpha_1e-05	0.62685	-0.086819	0.16899	0.54027	0.35784
alpha_0.0001	0.62685	-0.085848	0.16934	0.53916	0.35829
alpha_0.001	0.62679	-0.076079	0.17294	0.52777	0.36301
alpha_0.01	0.62254	-0.0010584	0.20467	0.45843	0.36975
alpha_0.1	0.48066	0.17818	0.37807	0.24243	0.22399
alpha_1	0.016549	-0.114	0.59399	0.0071774	0.004758
alpha_5	0	-0.12963	0.59971	0	0
alpha_10	0	-0.12963	0.59971	0	0
alpha_15	0	-0.12963	0.59971	0	0

	Model	training_score	testing_score	intercept	coef_x_1	coef_x_2
alpha_0.1	Elastic_model	0.48066	0.17818	0.37807	0.24243	0.22399

5. Linear Regression using Gradient Descent aka lr_scratch

	training_score	testing_score	intercept	coef_x_1	coef_x_2
learning_rate_1e-15	-4.43	-10.58	5.997e-12	3.581e-12	3.234e-12
learning_rate_1e-10	-4.43	-10.58	5.997e-07	3.581e-07	3.234e-07
learning_rate_1e-08	-4.429	-10.57	6.057e-05	3.616e-05	3.266e-05
learning_rate_1e-05	-3.167	-7.882	0.05563	0.03343	0.0302
learning_rate_0.0001	0.297	-0.31	0.3136	0.2038	0.1846
learning_rate_0.001	0.5986	0.1062	0.268	0.3735	0.3413
learning_rate_0.01	0.6266	-0.08031	0.1706	0.5069	0.3917
learning_rate_0.1	0.6269	-0.08691	0.169	0.5403	0.3578
learning_rate_0.5	0.6269	-0.08691	0.169	0.5404	0.3578
learning_rate_0.7	0.6269	-0.08691	0.169	0.5404	0.3578
learning_rate_1	0.6269	-0.08691	0.169	0.5404	0.3578

	Model	training_score	testing_score	intercept	coef_x_1	coef_x_2
learning_rate_0.001	lr_scratch	0.5986	0.1062	0.268	0.3735	0.3413