LINEAR AND LOGISTIC REGRESSION FOR PREDICTING THE GPU RUNTIME

ASSIGNMENT -1

NIDHI
NXX190003

INTRODUCTION: This report is about the implementation of linear regression on a given dataset by applying gradient descent to minimize the error and predict the response variable as accurate as possible. We are going to use SGEMM GPU kernel performance Data Set dataset available for download here.

DATA:

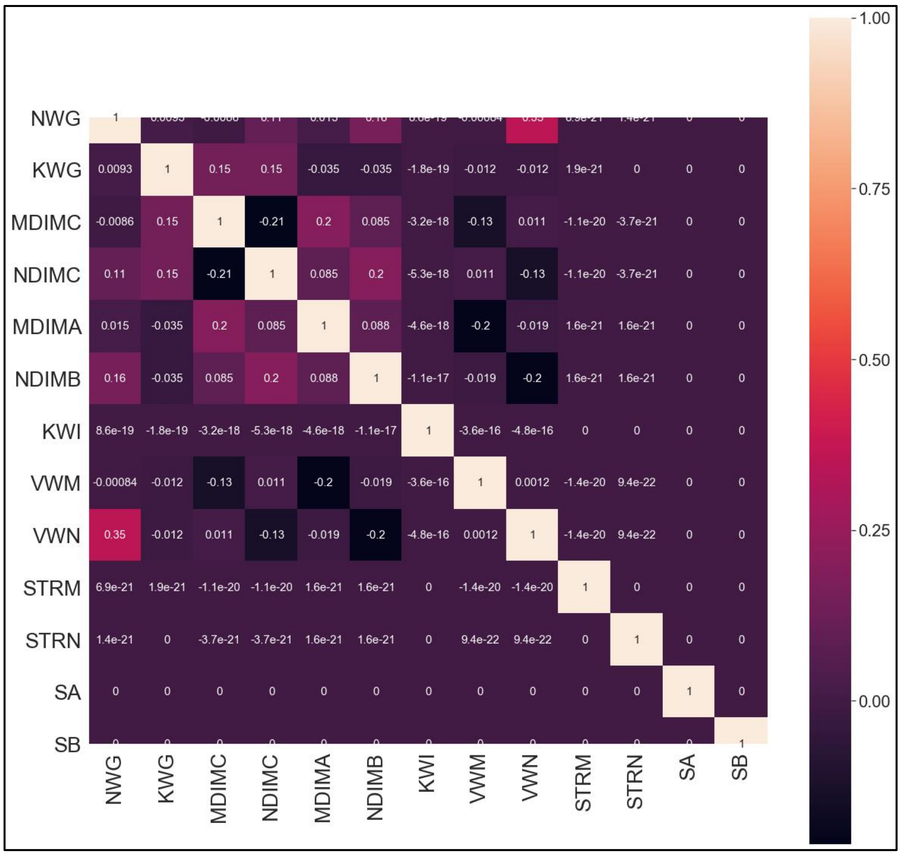
1)Dataset consists of 14 variables. First 10 variables are ordinal and last 4 variables are binary.

2)This data set measures the running time of a matrix-matrix product A*B = C, where all matrices have size 2048 x 2048, using a parameterizable SGEMM GPU kernel with 241600 possible parameter combinations.

3)For each tested combination, 4 runs were performed and their results are reported as the 4 last columns. All times are measured in milliseconds. This is a Multiple Linear Regression problem since it has multiple features (variables).

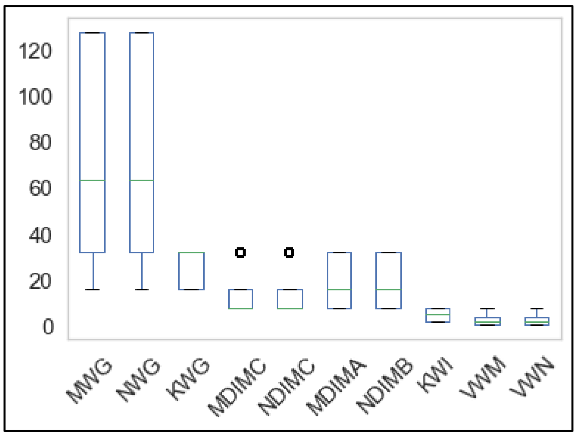
Exploratory Data Analysis – EDA

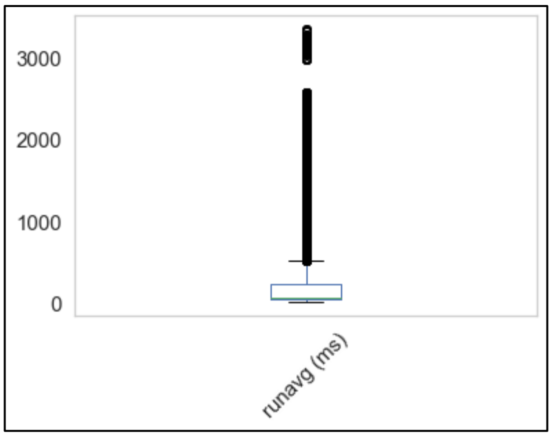
The fundamental target of performing EDA is to get an idea of the data before-hand. For this I have plotted heatmaps and pair plots to understand the overall relationship among different dependent variables.



By looking at the correlation plot, I find that there is no as such strong correlation between dependent variables.

Boxplot for Dependent and independent variables:





MWG and MWG ranges from 16 to 128. MDIMC and NDIMC. Maximum value of MDIMC and NDIMC is 8. STRM,STRN,SA and SB are not included in the box plot because they take only two values i.e. 0 and 1.

Second boxplot is of dependent variable runavg(ms) which is the average value of all runs. There are few outliers that's why while doing the logistic regression median is taken for the classification and not the mean.

Model Representation

Linear Regression:

Like Simple Linear Regression, we have input variable(X) and output variable(Y). But the input variable has n features. Therefore, we can represent this linear model in general form as follows;

Y = θoXo + θ1x1 + θ2x2 + ⋯ + θnxn

Where xn is the ith feature in input variable. By introducing X0 = 1, we can write this equation.

Y = θo + θ1x1 + θ2x2 + ⋯ + θnxn

And converting it to matrix form has its own benefits in Python.

Y = θTX

We must define the cost of the model. The cost function basically gives the error in our model. Y in above equation is our hypothesis(approximation) and we are going to define it as our hypothesis function as follows;

hθ(x)= θTX

And the cost is,

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2,$$

Gradient Descent

Gradient Descent is an optimization algorithm. We will optimize our cost function using Gradient Descent Algorithm becomes small enough (i.e. convergence takes place).

Step 1

Randomly generate thetas

Step 2

Iteratively update until it converges,

$$\theta := \theta - \alpha \frac{\delta}{\delta\theta} J(\theta).$$

Here α is the learning rate and [∂/∂βj] J(β) means we are finding partial derivate of cost w.r.t each βj. In step2, values of βj are changed each time in a direction such that our cost function is reduced. Gradient Descent gives the direction in which we want to move. As the model iterates, it gradually converges towards a minimum where further tweaks to the parameters produce little or zero changes in the loss — also referred to as convergence. We need learning rate because we don’t want to change values of βj drastically, because we might miss the minima.

Algorithm Preparation:

The steps followed are as follows:

1. Import the dataset into the workspace using the pandas read csv function.

2. Matrix of all the 14 independent variables is created.

3. Random matrix for theta is generated.

4. A column named runavg is created which is the average of all the four runtimes.

5. Dataset is split in the ratio of 70:30 in train and test datasets respectively and normalised.

7. Cost and gradient descent functions are setup and implemented.

8. Alpha is the learning rate which determines the size of step we are taking towards our cost function goal. Different alpha will have different effect on the result.

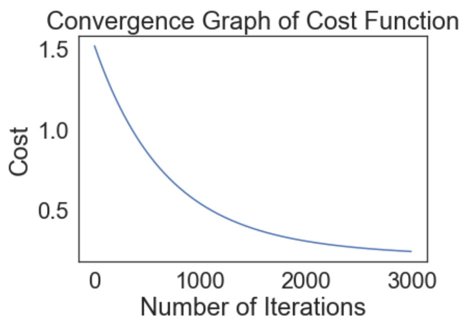
9. For Logistic Regression, the regression problem was converted into a classification problem by classifying all values above the median as 1 and value below as 0. Median is chosen because in case of outlier median gives the best result.

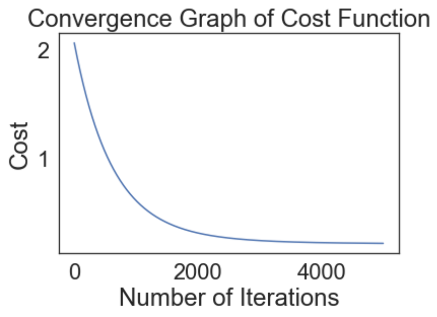
Experimentation:

Experiment 1: Experiment with various values of learning rate ∝ and report on your findings as how the error varies for train and test sets with varying ∝. Plot the results. Report your best ∝ and why you picked it.

Linear regression:

Case 1: Keeping Learning rate (alpha)= 0.001 and no of iterations = 3000 and 5000 respectively



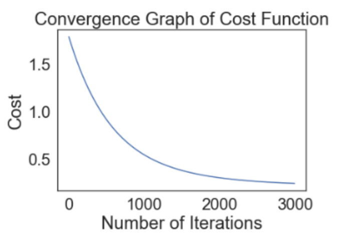


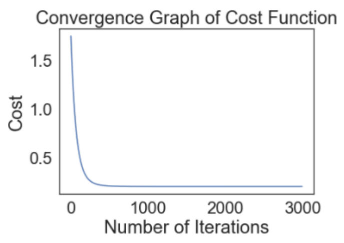
We see that for iterations=3000, cost is 0.248 and for iteration= 5000, cost is 0.208

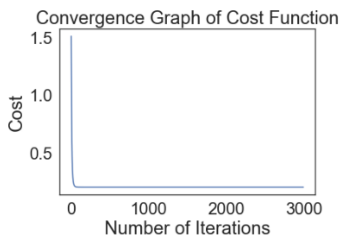
On increasing the number of iterations, value of the cost function decreases.

Case 2: Keeping iterations = 3000 and learning rate(alpha) = 0.001 ,0.01 and 0.1

Alpha	Iterations	Cost	Converging point	MSE train	MSE test
0.001	3000	0.234	2999	0.6892	0.6929
0.01	3000	0.207	900	0.5931	0.5912
0.1	3000	0.207	90	0.5931	0.5912





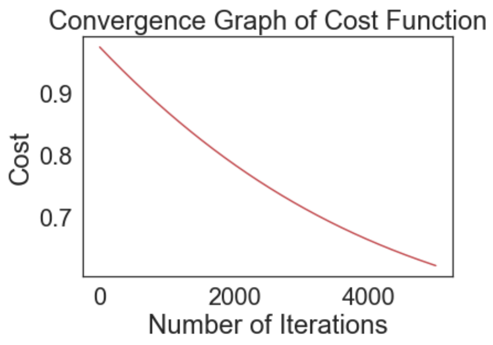


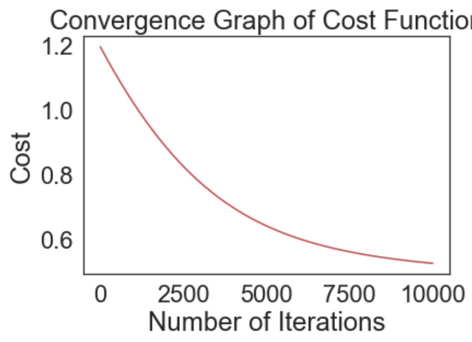
Result: The cost function value at alpha = 0.01 and 0.1 are same i.e. 0.207. Hence, taking alpha value of 0.1 looks more promising because it takes adequate steps (90) to reach the global minimum assuring more accurate value when number of iterations will be increased. With alpha value higher, there is a possibility that the cost function may shoot the global minimum value and keep iterating in the loop. Therefore, I have chosen the alpha = 0.1.

Run= 0.82+ 0.380*MWG+ 0.354 *NWG+0.112 *KWG -0.357* MDIMC-0.349 *NDIMC-0.027 *MDIMA+0.028* NDIMB+0.032* KWI-0.006* VWM-0.017* VWN-0.018* STRM+0.00005* STRN+0.052* SA+0.064* SB

Logistic Regression:

Case 1: Keeping Learning rate (alpha)= 0.001 and no of iterations = 5000 and 10000 respectively

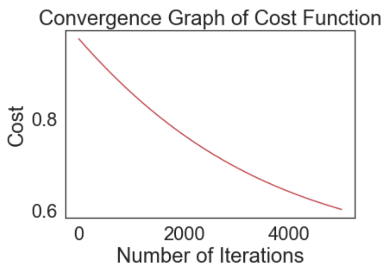


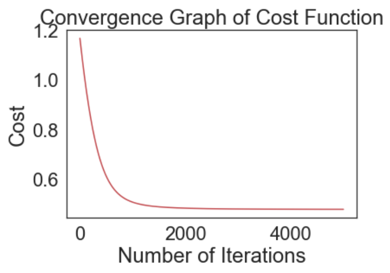


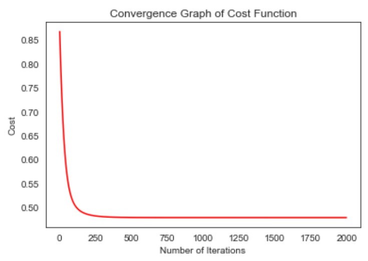
On increasing the number of iterations, value of the cost function decreases.

Case 2: Keeping iterations = 5000 and learning rate(alpha) = 0.001 ,0.01 and 0.1

Alpha	Iterations	Cost	Converging point	Accuracy
0.001	5000	0.610	4999	71.4%
0.01	5000	0.478	4000	81.2%
0.1	5000	0.478	500	81.4%







The cost function value at alpha = 0.01 and 0.1 are approximately similar. Hence, taking alpha value of 0.1 looks more promising because it takes adequate steps (500) to reach the global minimum assuring more accurate value when number of iterations will be increased. Also, the accuracy for alpha=0.1 is highest being 81.4%. Therefore, I have chosen the alpha = 0.5.

Run= 0.954+1.30* MWG + 0.84* NWG + 0.15* KWG -0.89* MDIMC -0.76* NDIMC -0.04* MDIMA -0.007* NDIMB -0.02* KWI -0.078* VWM -0.15* VWN-0.32* STRM-0.02* STRN+0.40* SA+0.08* SB

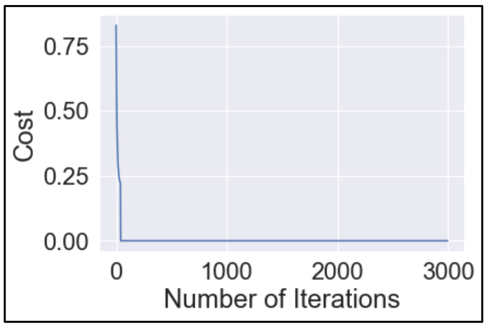
Experiment 2:

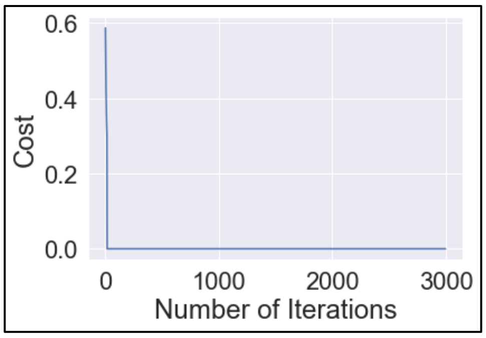
Experiment with various thresholds for convergence for linear and logistic regression. Plot error results for train and test sets as a function of threshold and describe how varying the threshold affects error. Pick your best threshold and plot train and test error (in one figure) as a function of number of gradient descent iterations.

The threshold value is an important parameter because it defines when to stop computing the cost function it denotes that the convergence has been achieved while training the data. It doesn’t let the unnecessary iterations takes place if the convergence has been achieved or likely to achieve earlier in the iteration.

Here I have taken two threshold values for both Linear and Logistic Regression: 0.001 and 0.01.

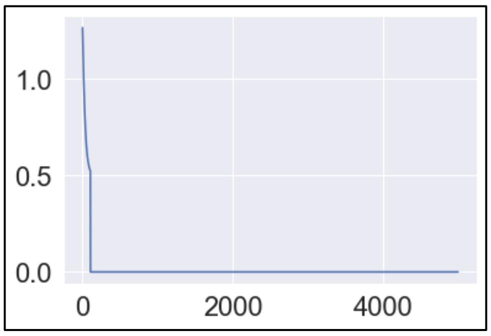
Linear:

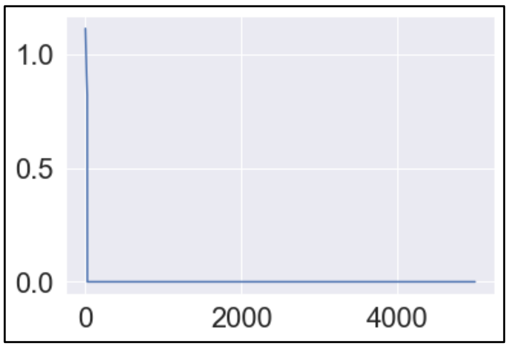




For threshold value 0.001, 39 steps are taken to converge and for 0.01 13 steps are taken.

Logistic:





For threshold value 0.001, 104 steps are taken to converge and for 0.01 23 steps are taken.

Result: Keeping the threshold values higher results into forcing the convergence to happen at an earlier stage which could be a drawback. Here, I am choosing 0.01 as my threshold value.

Experiment 3:

Pick eight features randomly and retrain your models only on these ten features. Compare train and test error results for the case of using your original set of features (14) and eight random features. Report the ten randomly selected features.

In this experiment I have picked the following 8 features randomly:

MWG, KWG, NDIMC, NDIMB, VWM, STRM, STRN, SA

Rest of the 6 features are dropped. Model equation for this experiment is:

Run = βo+β1*MWG + β2*KWG + β3* NDIMC + β4*NDIMB + β5*VWM + β6*STRM + β7*STRN + β8*SA

For Linear Regression:

The minimum value of cost function for original model is 0.207 whereas the minimum value of cost function for this experiment model is 0.288

Comparing the MSE value of the original model and model of this experiment,

MSE for original model= 0.591

MSE for this experiment model = 0.823

MSE for experimental model has increased as compared to the original model.

For Logistic Regression:

The minimum value of cost function for original model is 0.478 whereas the minimum value of cost function for this experiment model is 0.5744

Comparing the accuracy of the original model and model of this experiment,

Accuracy of original model= 81.4%

Accuracy of experiment model = 73.0%

Result:

I see that when the features are randomly selected, MSE and inaccuracy increases as compared to the original model.

Experiment 4:

Now pick eight features that you think are best suited to predict the output, and retrain your models using these ten features. Compare to the case of using your original set of features and to the random features case. Did your choice of features provide better results than picking random features? Why? Did your choice of features provide better results than using all features? Why?

In this experiment I have picked the following 8 features.

MWG, NWG, MDIMC, NDIMC, VWM, VWN, SA and SB

These 8 features are picked depending on the correlation with the dependent variable, in this case runavg(ms). All these 8 features have high correlation with the dependent variable.

Model equation:

Run = βo+β1*MWG + β2*NWG + β3* MDIMC + β4*NDIMC + β5*VWM + β6*VWN + β7*SA + β8*SB

Comparing the MSE value of the following:

Original model= 0.591

Experiment model 3= 0.823

Experiment model 4= 0.604

Observation: MSE for experimental model is approximately similar the original model.

For Logistic Regression:

Comparing the accuracy of the following:

Original model= 81.4%

Experiment model 3= 73.0%

Experiment model 4= 80.8%

Observation: Accuracy for experimental model is approximately similar to the original model.

Result: If I compare the result, I find out that the choice of the feature selection turned out to be a good decision as compared to the choice when variables were randomly picked. The reason being that these 8 features have a good correlation factor with the target variable and these features are depicting behaviour about the target variable.

Discussion:

After performing the four experiments following are the insights:

1) Learning rate and threshold are the important parameters to be kept in mind while performing both linear and logistic regression. Very high learning rate and threshold may force the convergence to happen at an earlier stage which could be a drawback.
2) Model based on random selection of features provides misleading result. So, it is advised to choose features keeping some factors in mind. Ex-correlation with the target variable.
3) All the 10 variables in the dataset is treated as continuous variable here. The result and model performance could have been better if we add weights and encode the values in these columns.
4) Value of R-square in linear model is 0.40. There is a possibility that better insights on the model performance could be drawn if the model is non-linear.