Prediction of SGEMM GPU Kernel Performance

Introduction:

The emergence of programmable graphics hardware has led to increasing interest in offloading numerically intensive computations to GPUs. This has made GPUs very prevalent in this era of high performance computing, but GPU programming remains challenging and it would therefore be handy to have a model which captures the main performance factors of GPU kernels that helps to estimate the run time, which could also potentially be used to identify its bottlenecks.

In this project, the objectives are to implement Linear and Logistic Regression using Batch Gradient Descent algorithm to predict the GPU run time. Additionally, experiments were conducted to identify the optimal learning rate, threshold values and the models with low error and high accuracy after feature selection.

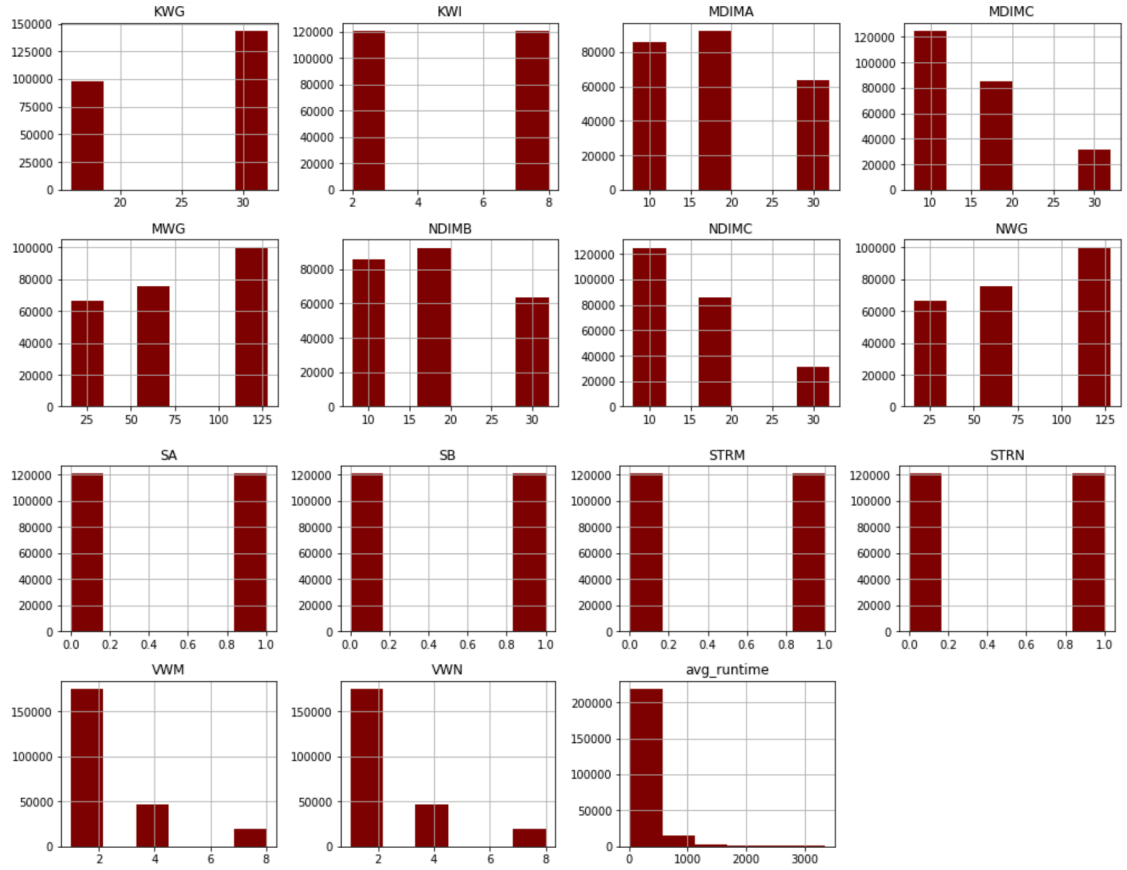
Dataset Description:

We will be using the SGEMM GPU kernel performance dataset. 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. For each tested combination, 4 runs were performed, and their results are reported as the 4 last columns.

It has **4 dependent** variables and **14 independent** variables - the first 10 are ordinal and can only take up to 4 different powers of two values, and the 4 last variables are binary.

Exploratory Data Analysis:

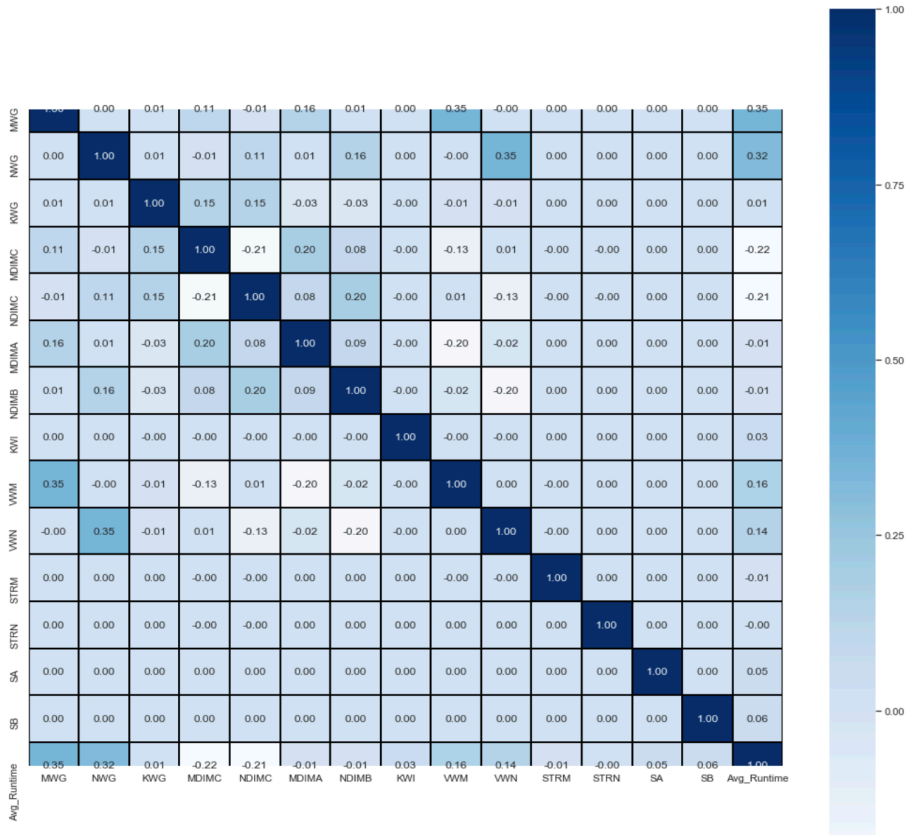
Before jumping in with the implementation of ML algorithm on your data, it is necessary to completely understand the given dataset and check for correlation among variables, skewness, outliers, inconsistent and missing values to build an efficient model. For this purpose, histograms of all the parameters and correlation matrix are plotted.



Feature Distribution Plots

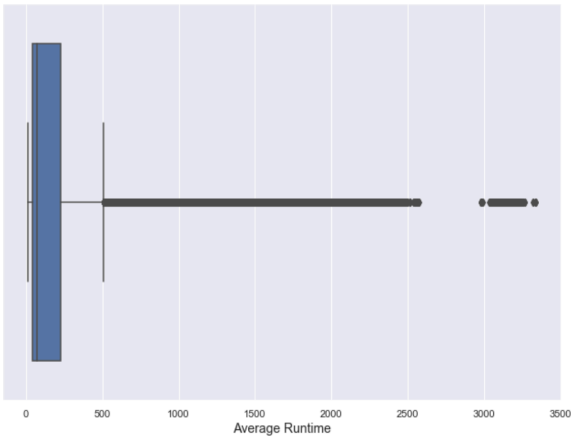
***Missing Values*:** No missing values in this dataset

***Correlation*:** Since this dataset consist of records of all possible combination of the 14 parameters, it is highly unlikely that the variables will be correlated. This was confirmed from correlation matrix that there exists no significant correlation between the variables.



Correlation Matrix

***Skewness*:** When you look at the distribution plot for the average run time (avg_runtime), it can be seen that it is positively skewed. On further analysis, it can be seen that, the 75th percentile of the average runtime is 228ms while its maximum value is 3341ms. This shows there could be **outliers** in this data which might affect the model. This is further justified with the help of the adjacent box plot:



To address this, I have made an assumption that the avg_runtime follows Gaussian Distribution and **considered only records till 2sigma for our analysis**.

Though it can be argued that the run time is measured 4 times and average of it is taken which eliminates the possibility of measurement errors, there could be other factors causing this. And

it is therefore safe to consider the 2sigma approach as we are just eliminating 4% of the data which could possibly be outliers.

Data Preparation:

It is important to **scale** the features to a range which is centered around zero i.e. to **have 0 mean and 1 SD**. This is done so that the standard deviation of the features is in the same range. If a feature's variance is orders of magnitude more than the variance of other features, that particular feature might dominate other features in the dataset. To address this, **the data has been normalized by subtracting each value with its mean and divide by its standard deviation.**

Experimentation:

Experiment 1 & 2

Linear Regression:

Firstly, I have split the dataset into Training and Test set in the ratio of **70:30**.

Linear Regression was then implemented using the **gradient descent algorithm** and the best model was identified by varying the hyperparameter **alpha** and **threshold** values.

Here are the results:

Alpha	Loss	Iterations	RMSE Train	RMSE Test	R-Squared	[2] Wall time (sec)
0.001	12783.82	27847	159.89	159.46	0.488	54.6
0.01	12783.82	3017	159.89	159.46	0.488	5.89
0.1	12783.82	327	159.89	159.46	0.488	0.957
1	12783.82	31	159.89	159.46	0.488	0.301
1.11	12783.82	36	159.89	159.46	0.488	0.34
1.4	12783.82	1191	159.89	159.46	0.488	2.47
1 / (1 + iteration)	12783.89	[1] 30000+	159.89	159.46	0.488	62.5

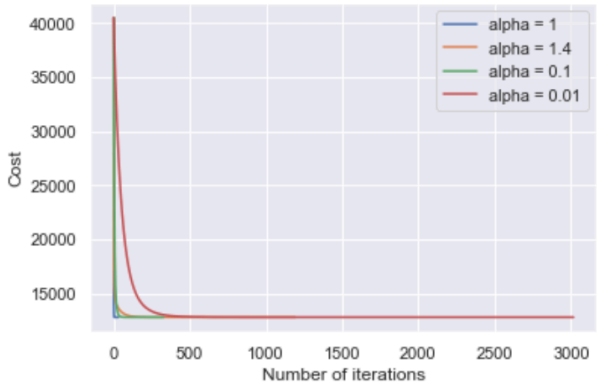
[1] Loss dint converge even at 30000 iteration with a threshold value of 0.000001

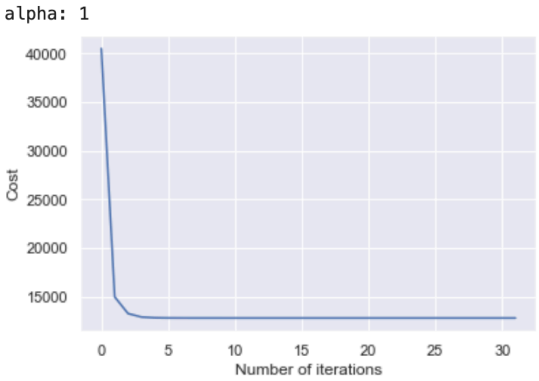
[2] Wall time will vary for each run. But it is always around the above-mentioned values.

Interpretation of the above results: Here the RMSE value is **159.89**. For the given dataset, the standard deviation of the average runtime variable is 223. Hence, RMSE value of 159.89 is not as bad as it seems to be.

And in case of R-Squared, though **0.488** is a low value, it is clearly seen that the distribution of the target variable is not a straight line. Here, in this project we are just implementing simple linear regression (no higher order polynomials used). Hence, R-Squared value of 0.488 seems reasonable. Had we removed all the records above the 75th percentile, R-Squared would have been significantly higher and RMSE would have been lower. But then we cannot completely justify removal of 1/4th of the records.

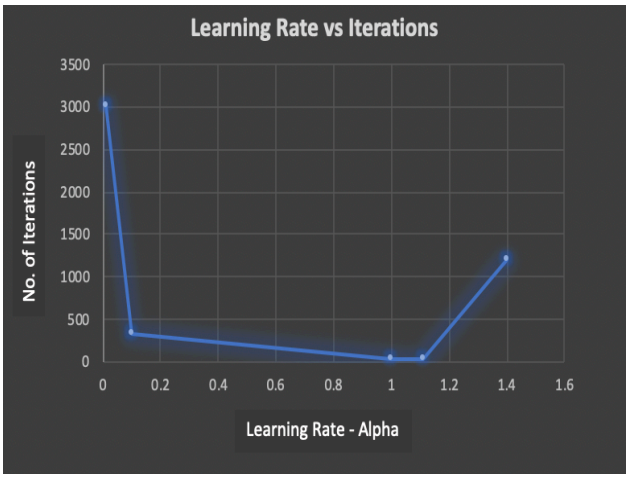
The regression was run for 7 different alpha values – 0.001, 0.01, 0.1, 1, 1.11, 1.4, 1/(1+iteration) (i.e. variable step-size) at a threshold value of 0 and its corresponding Loss, Iterations, RMSE – Train & Test, R-Squared and Wall time were calculated.
Following are the Loss/Cost plots:





Please note that loss curve for all the above-mentioned alpha values are available in the python code attachment.

From the above graphs and table, alpha values between 0.1 to 1.11 were found to be ideal range for the linear regression on this dataset.
And, it can be clearly seen that **alpha = 1** is the optimal value where the linear regression performs the best since it takes only 31 iterations to converge and that too within 0.301 seconds.



Setting the alpha value as 1, now the threshold values were varied. It was noticed that increasing the threshold from 0, only increased loss value. Though the number of iterations

were slightly reduced, this doesn't much change the time taken which was already low at 301ms. It is therefore good to set the threshold value as 0 rather than increasing it.

The resulting regression equation from the model using the alpha value as 1 and threshold as 0 is:

avg_runtime = -7.09 – 2.34 (MWG) – 1.82 (NWG) – 1.37 (KWG) + 2.37 (MDIMC) + 2.23 (NDIMC) + 3.54 (MDIMA) + 2.49 (NDIMB) – 4.0 (KWI) + 2.27 (VWM) + 1.76 (VWN) + 2.74 (STRM) – 1.87 (STRN) – 9.05 (SA) – 7.94 (SB)

Logistic Regression:

Here, the target value must be considered as a binary value since logistic regression is run for variable classification. This is done by considering all the values of average runtime **above** the **median as 1** (this means computation takes more time) and **below the median as 0** (this means computation takes lesser time).

This dataset consists of all the possible combination of GPU kernel parameters. Since it represents the complete set of data for the given GPU configuration, it is safe to assume that anything above the median value as high time consuming.

Similar to Linear Regression, Logistic Regression was implemented using the gradient descent algorithm and the best model was identified by varying the hyperparameter alpha and threshold values.

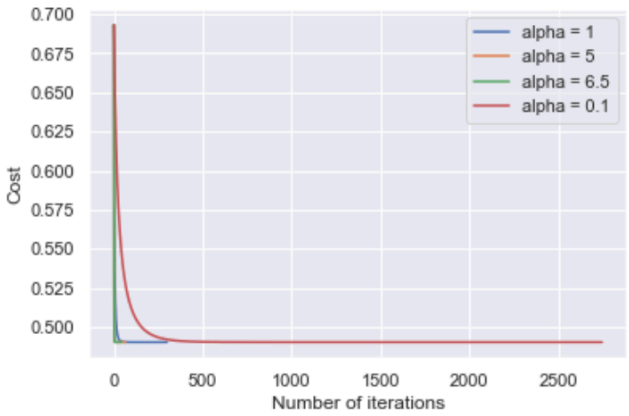
Here are the results:

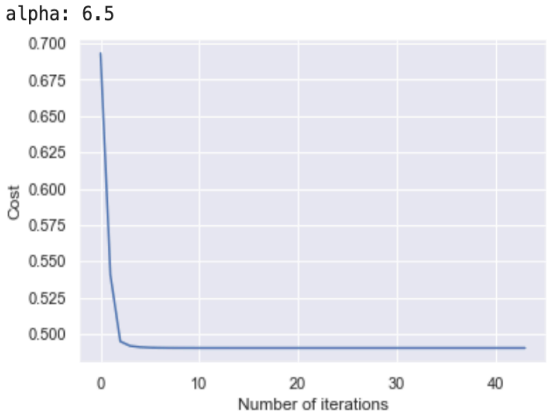
Alpha	Loss	Iterations	Accuracy Train	Accuracy Test	[1] Wall time (sec)
0.001	0.4903	56825	0.799	0.8	373
0.01	0.4903	24986	0.799	0.8	173
0.1	0.4903	2747	0.799	0.8	17.8
1	0.4903	299	0.799	0.8	2.34
5	0.4903	66	0.799	0.8	0.733
6.5	0.4903	43	0.799	0.8	0.535

[1] Wall time will vary for each run. But it is always around the above-mentioned values.

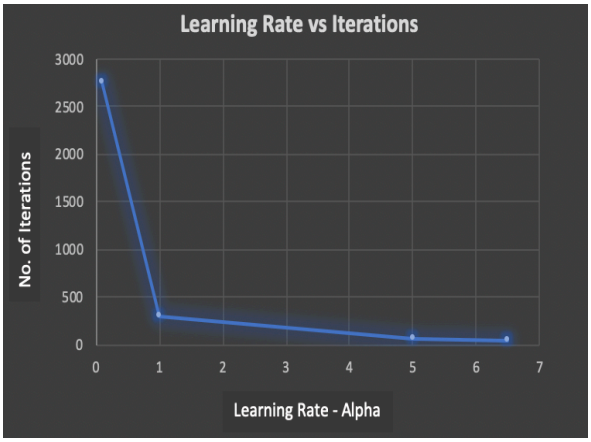
The regression was run for 6 different alpha values – 0.001, 0.01, 0.1, 1, 5, 6.5 at a threshold value of 0 and its corresponding Loss, Iterations, Accuracy – Train & Test and Wall time were calculated.

Following are the Loss/Cost plots:





Please note that loss curve for all the above-mentioned alpha values are available in the python code attachment.



From the above graphs and table, alpha values between 1 to 6.5 were found to be ideal range for the logistic regression on this dataset.

And, it can be clearly seen that **alpha = 6.5** is the optimal value where the linear regression performs the best since it takes only 43 iterations to converge and that too within 0.535 seconds.

Setting the alpha value as 6.5, now the threshold values were varied. It was noticed that increasing the threshold from 0, only increased loss value. Though the number of iterations were slightly reduced, this doesn't much change the time taken which was already low at 535ms. It is therefore good to set the threshold value as 0 rather than increasing it.

The resulting regression equation and confusion matrix from the model using the alpha value as 6.5 and threshold as 0 is:

```
Confusion Matrix:
[[27451  6925]
 [ 6841 27639]]
Classification Report
              precision    recall  f1-score   support

           0       0.80      0.80      0.80     34376
           1       0.80      0.80      0.80     34480

    accuracy                           0.80     68856
   macro avg       0.80      0.80      0.80     68856
weighted avg       0.80      0.80      0.80     68856
```

log(avg_runtime) = – 3.44 (MWG) – 5.64 (NWG) – 1.23 (KWG) + 5.13 (MDIMC) + 4.71 (NDIMC) + 1.01 (MDIMA) + 7.55 (NDIMB) + 1.4 (KWI) + 3.79 (VWM) + 1.81 (VWN) + 1.20 (STRM) + 1.03 (STRN) + 1.53 (SA) + 2.35 (SB)

Experiment 3

10 features were randomly selected, and a linear regression model was built using the best alpha and threshold values selected before.

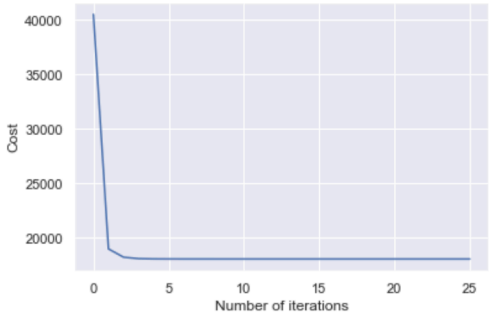
avg_runtime = b0 + b1 (MWG) + b2 (KWG) + b3 (MDIMC) + b4 (MDIMA) + b5 (KWI) + b6 (VWN) + b7 (STRN) + b8 (SA) + b9 (SB) + b10 (VWM)

The beta coefficients of the linear regression model and loss plot are as follows:

```
Loss Function Converges at 25

Model Loss:  18041.108062200186

Model Weights:
[[-7.74003173e-07]
 [-2.01665798e-08]
 [ 2.04296767e-07]
 [ 4.79376869e-07]
 [-4.20598685e-09]
 [ 6.06790541e-09]
 [ 8.56485689e-10]
 [-6.18138491e-09]
 [-6.81515999e-09]
 [ 8.05665760e-07]]
Model Bias:
-7.574216445328e-15
```




```
CPU times: user 920 ms, sys: 57.7 ms, total: 978 ms
Wall time: 378 ms
```

The Train and Test errors were calculated, and it was compared with the model containing all the features.

Models	RMSE Test	RMSE Train	R-Squared
All Features	159.9	159.9	0.488
10 Features	189.65	189.65	0.276

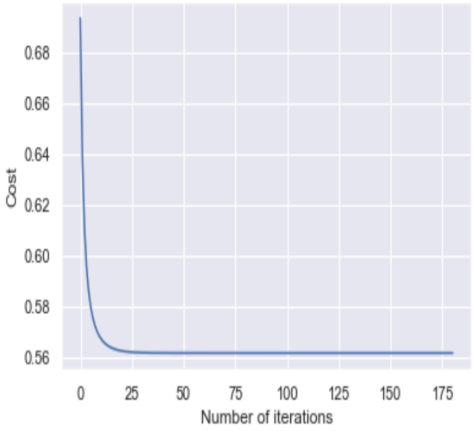
On random feature selection, R-Squared value decreased by 0.212 and RMSE value decreased by 18.6%**.** So, it is not definitely not as good as the model with all the features.

The beta coefficients of the logistic regression model and loss plot are as follows:

```
Loss Function Converges at 180

Model Loss:  0.5618043440732273

Model Weights:
[[-9.67727283e-09]
 [-5.67449911e-10]
 [ 5.08088811e-09]
 [ 4.33155631e-09]
 [ 9.98601114e-12]
 [-1.02185487e-09]
 [ 1.18297385e-10]
 [ 1.35252850e-09]
 [ 2.77919466e-10]
 [ 7.74585819e-09]]
alpha: 1
```



The accuracy now is 72% which is 8% less than the model with all the features.

Experiment 4

Now, 10 features were chosen in such a way that its beta coefficients are highly significant. Theoretically, its variable coefficients should have low standard error and high t-stat value.

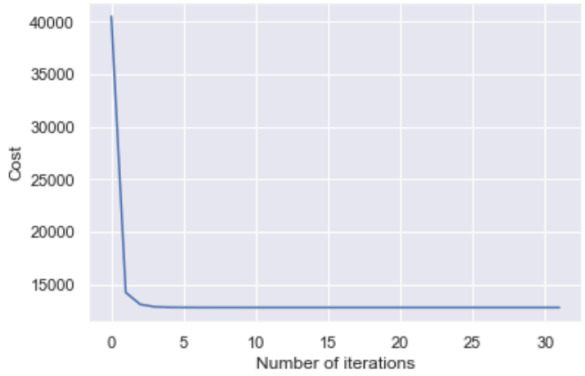
avg_runtime = b0 + b1 (MWG) + b2 (NWG) + b3 (KWG) + b4 (MDIMC) + b5 (NDIMC) + b6 (MDIMA) + b7 (NDIMB) + b8 (VWN) + b9 (STRM) + b10 (SA)

The beta coefficients of the linear regression model and loss plot are as follows:

```
Loss Function Converges at 31

Model Loss:  12785.407297261758

Model Weights:
[[-1.41233800e-09]
 [-1.81272873e-08]
 [-9.09566875e-09]
 [ 1.23204771e-08]
 [ 1.64172673e-08]
 [-7.33100894e-09]
 [ 5.20289675e-09]
 [ 1.82103785e-08]
 [ 2.89715341e-10]
 [-5.03847425e-10]]
Model Bias:
-3.453118110351331e-15
```



Following are the RMSE, R-Squared values for all the 3 models:

Models	RMSE Test	RMSE Train	R-Squared
All Features	159.9	159.5	0.488
10 Random Features	189.65	189.65	0.276
Best 10 Features	159.8	159.4	0.488

The Best 10 features selected now is definitely better than randomly selected features. This can be seen in the above table. And it is close to that of all the features since we have selected the best 10 features that is mainly contributing in the prediction of the target variable average runtime. Similar results were obtained from implementation of logistic regression as well.

Discussions:

- Describe your interpretation of the results.
  Explained in Experiment 1 & 2

- What do you think matters the most for predicting the GPU run time?
  Data Cleaning is very important before developing any ML model.
  In this case, removal of outliers was very important.
  With respect to the variables, apart from per-matrix widths and per-matrix manual caching, all other variables matter in predicting the GPU runtime i.e. "MWG","NWG","KWG","MDIMC","NDIMC","MDIMA","NDIMB" and "STRM".

- What other steps you could have taken with regards to modeling to get better results?
  - A model with a higher order polynomial could have been a better fit to the given dataset.
  - Better ML techniques like Neural Network can definitely give better results.
  - Mapping to higher dimensions can give better classification of the target variables. Support Vector Machine could be used.