Average GPU runtime prediction

Introduction:

This data set measures the running time of a matrix * matrix product. For each tested combination 4 runs were performed and their results are reported in the last four columns. We need to calculate the average runtime of these 4 columns to create our analysis on predicting the average gpu runtime for each case.

The average gpu runtime prediction will come under supervised learning task aiming to predict gpu runtime prediction for a given matrix of size 2048 x 2048 with factors like MWG, NWG, KWG, MDIMC, NDIMC, MDIMA, NDIMB, KWI, VWM, VWN, STRM, STRN, SA, SB. Many techniques like gradient descent algorithm, linear regression, logistic regression (in-built function) have been applied to predict the average gpu runtime.

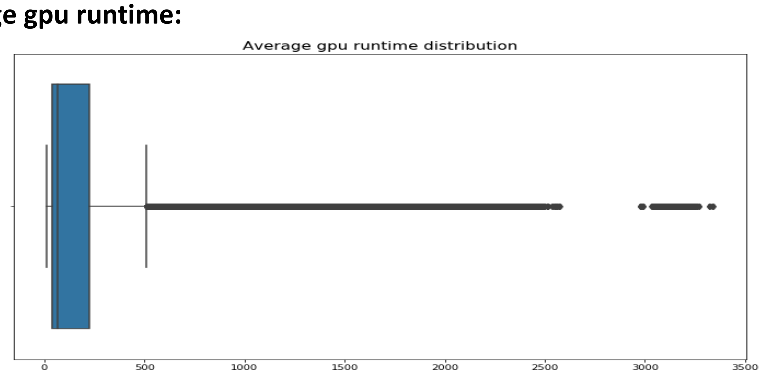
Dataset:

The dataset (SGEMM GPU kernel performance) dataset can be downloaded at:
https://archive.ics.uci.edu/ml/datasets/SGEMM+GPU+kernel+performance#
There are 14 parameters. The first 4 are ordinal and the last four variables are binary. The dataset has total 241600 data entries and 18 features with the last four being the runtime measurement.

Feature range:

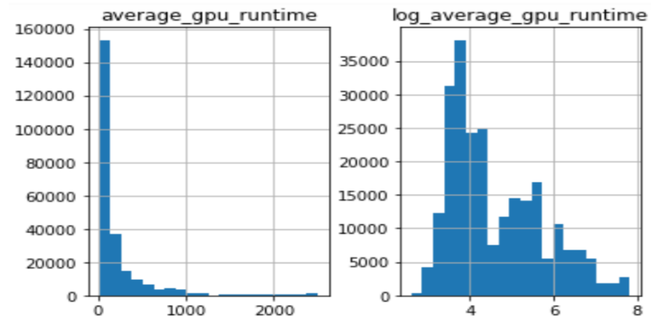
1. MWG, NWG : 16 to 128
2. KWG : 16 to 32
3. MDIMC, NDIMC, MDIMA, NDIMB : 8 to 32
4. KWI : 2 to 8
5. VWM, VWN : 1 to 8
6. STRM, STRN : 0 to 1
7. SA, SB : 0 to 1
8. Run1 (ms) : 13.29 to 3339.63 (milli second)
9. Run2 (ms) : 13.25 to 3375.42 (milli second)
10. Run3 (ms) : 13.36 to 3397.08 (milli second)
11. Run4 (ms) : 13.37 to 3361.71 (milli second)
12. Average_gpu_runtime : 13.31 to 3341.50 (milli second)

Outlier detection for average gpu runtime:



The major reading lies between 0 – 500 (milli second) range. The value of the 25th percentile is 40.66 milli second, 50th percentile is 69.79 milli second and 75th percentile is 228.38 milli second. The minimum value is 13.31 milli second and maximum value is 3341.5 milli second. The number of the 0.1% top values of average gpu runtime is 2500 milli second. So, we remove the instance above the 2500 milli second mark.

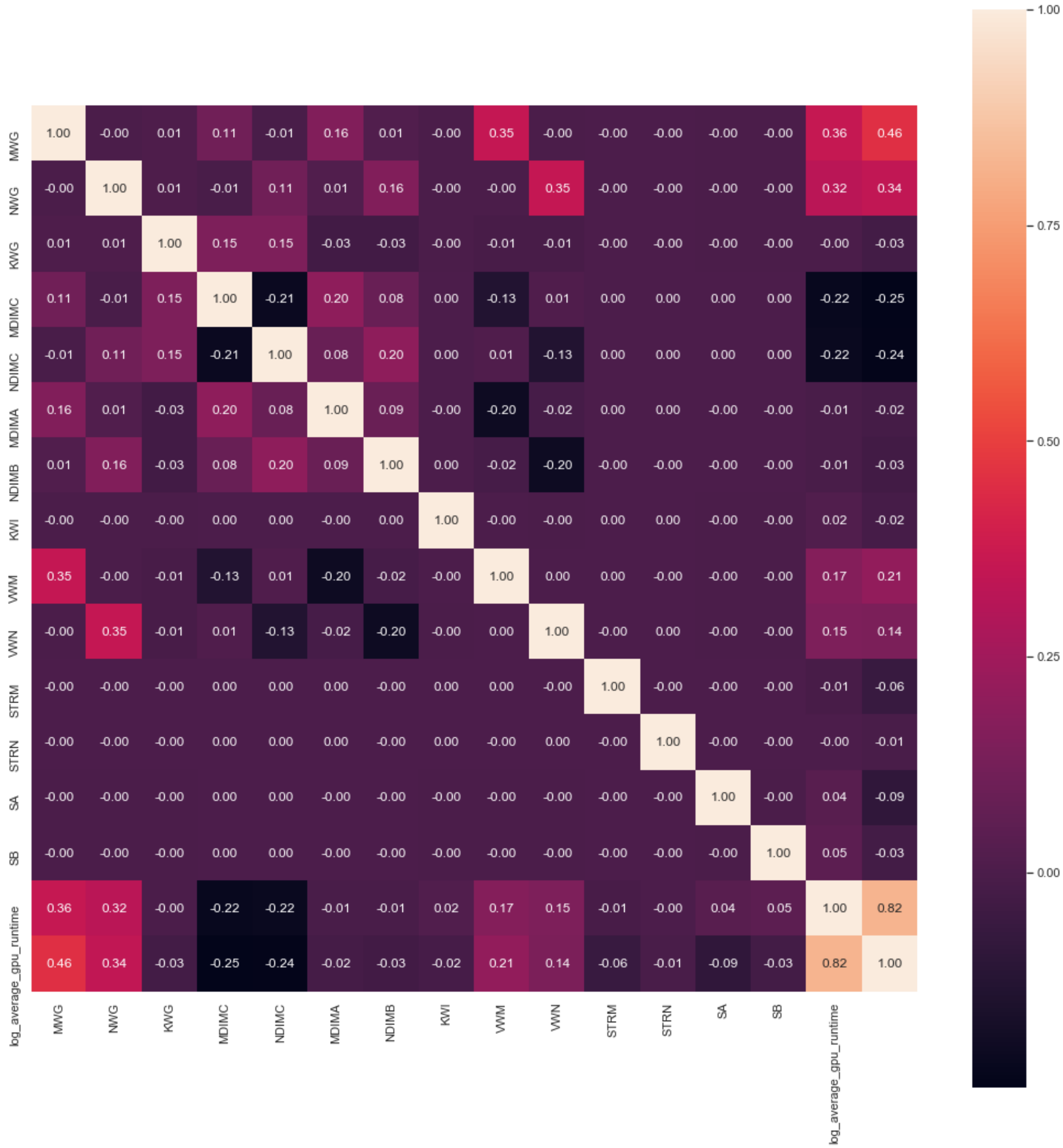
Adjusting the distribution:



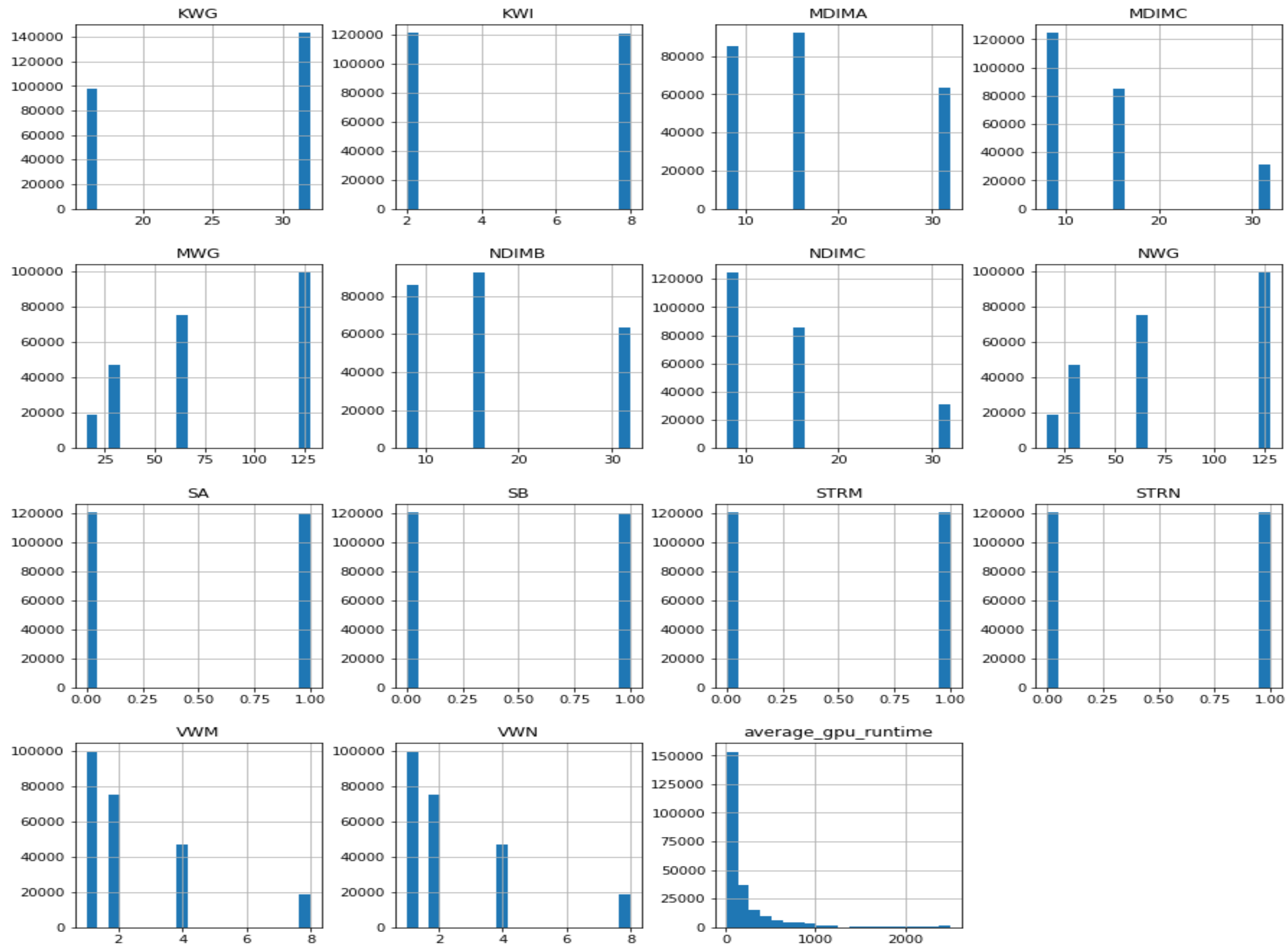
The distribution of average gpu runtime is not normal as we have left symmetry, for this reason I will use in my analysis log (average_gpu_runtime) which distribution is more normal.

Correalation:

The most correlated values with gpu runtime (log_average_gpu_runtime) are: MWG = 0.36, NWG = 0.32, VWM = 0.17, VWN = 0.15.



Feature distribution:

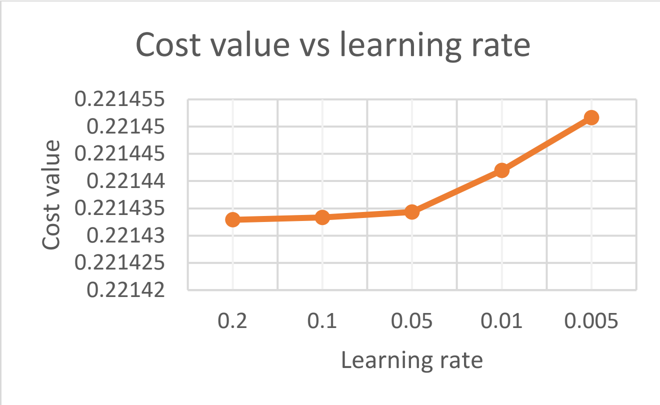


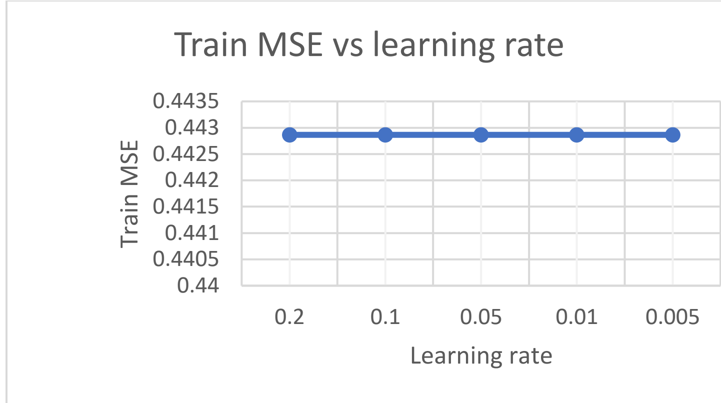
<u>Experiment 1:</u>

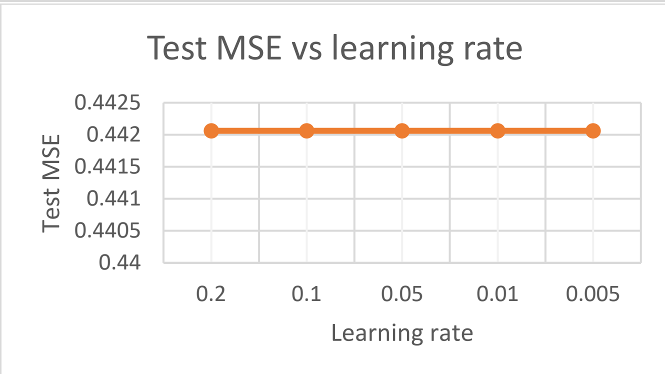
Model :
Average_gpu_runtime = b0 + (b1 * MWG) + (b2 * NWG) + (b3 * KWG) + (b4 * MDIMC) + (b5 * NDIMC) + (b6 * MDIMA) + (b7 * NDIMB) + (b8 * KWI) + (b9 * VWM) + (b10 * VWN) + (b11 * STRM) + (b12 * STRN) + (b13 * SA) + (b14 * SB)

Initial coefficients : - 0, initial alpha = 0.5, data split = 80:20.
I have fixed the alpha as “0.0000001” and varied learning rate 0.2, 0.1, 0.01, 0.05, 0.001, 0.005, 0.0001, 0.0005. To find the best alpha value I ran linear regression using the in-built functions and matched the coefficients.

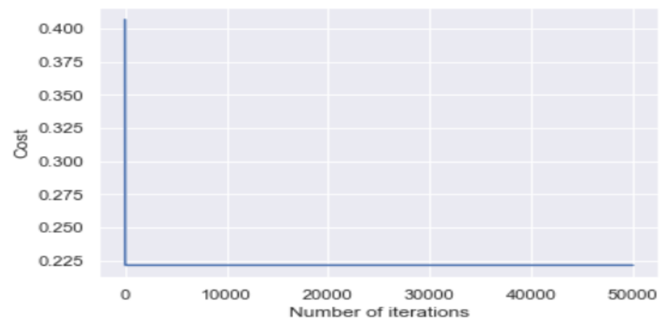
Regression model equation:
Average_gpu_runtime = 0.5020 + (0.3963 * MWG) + (0.0844 * NWG) + (-0.3959 * KWG) + (-0.3817 * MDIMC) + (0.0005 * NDIMC) + (-0.0012 * MDIMA) + (-0.0133 * NDIMB) + (-0.0154 * KWI) + (-0.0410 * VWM) + (-0.0607 * VWN) + (-0.008 * STRM) + (-0.0088 * STRN) + (-0.0883 * SA) + (-0.0247 * SB).



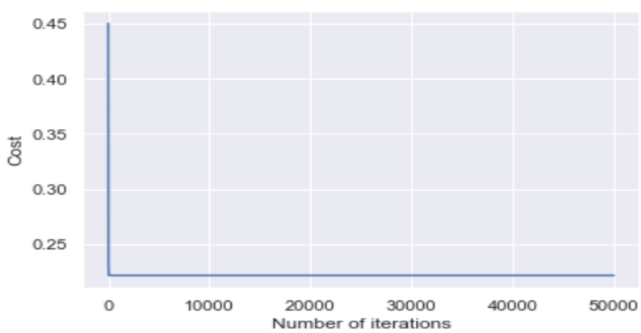




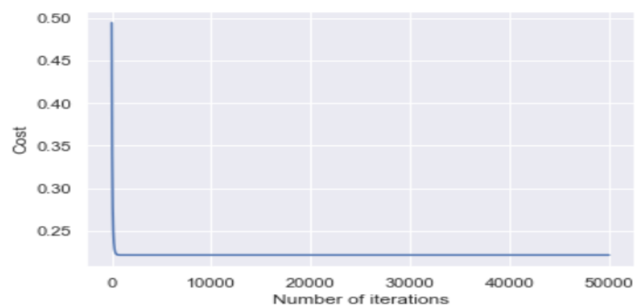
alpha: 0.2



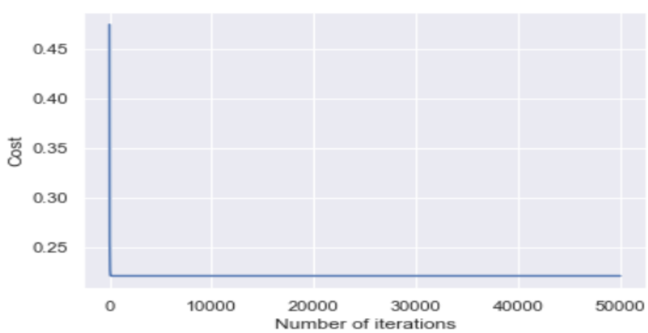
alpha: 0.1



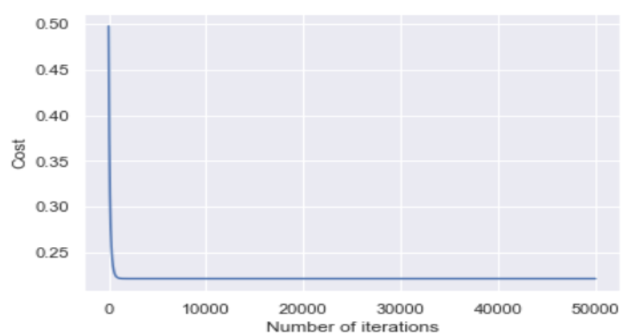
alpha: 0.01



alpha: 0.05



alpha: 0.005



Learning rate	Cost value	Converging point	Train MSE	Test MSE
0.2	0.221432894	56	0.44286479	0.44206195
0.1	0.221433359	110	0.44286479	0.44206195
0.05	0.221434302	211	0.44286479	0.44206195
0.01	0.221441975	915	0.44286479	0.44206195
0.005	0.221451653	1699	0.44286479	0.44206195

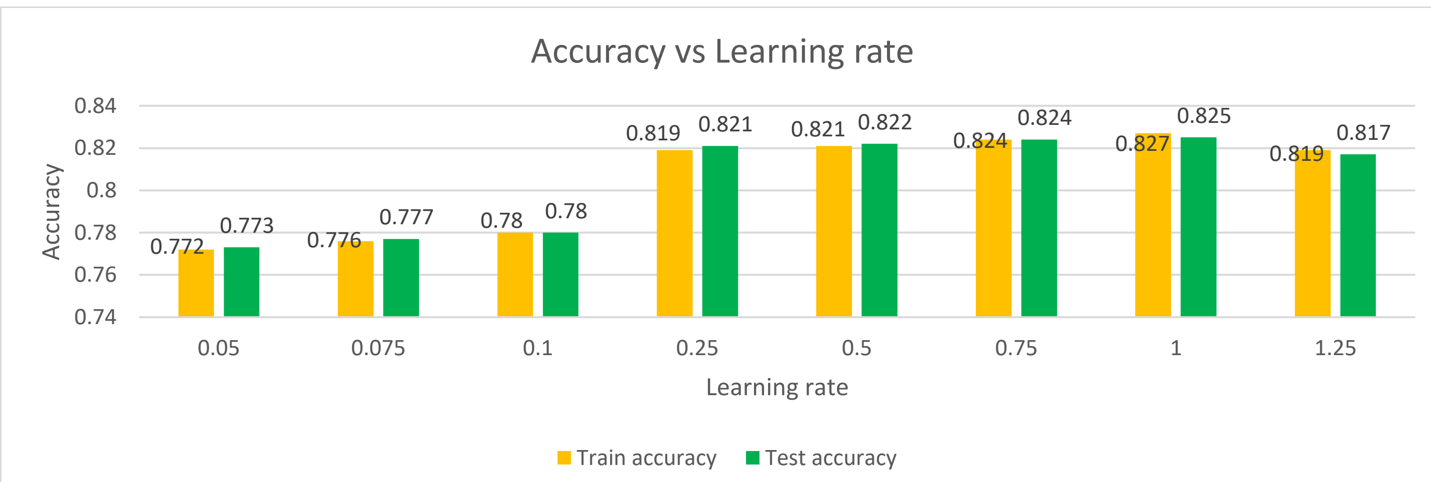
- The best value of alpha is 0.2 with cost value = 0.221432894 with the converging point = 56 train MSE value 0.44286479 and test MSE value 0.44206195.
- As alpha decreases the cost value and the converging point increases.

Logistic regression:

Class construction:

I created a new variable called average_gpu_runtime_class. It is a categorical variable and it takes "1" for average_gpu_runtime value greater than or equal **70 milli second (median)** and "0" for average_gpu_runtime value less than 70 milli second.

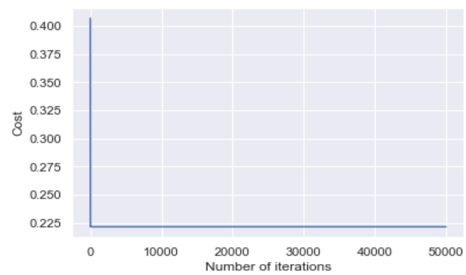
I used **gradient boosting classifier** to classify average gpu runtime class for different learning rates (0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1, 1.25)



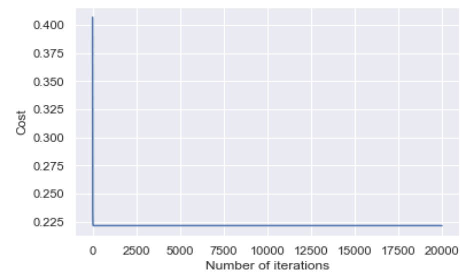
Learning rate	Train accuracy	Test accuracy	Precision	Recall	F1-Score
0.05	0.772	0.773	0.78	0.77	0.77
0.075	0.776	0.777	0.78	0.78	0.78
0.1	0.78	0.78	0.78	0.78	0.78
0.25	0.819	0.821	0.82	0.82	0.82
0.5	0.821	0.822	0.82	0.82	0.82
0.75	0.824	0.824	0.82	0.82	0.82
1	0.827	0.825	0.83	0.83	0.83
1.25	0.819	0.817	0.82	0.82	0.82

Experiment 2 – Changing threshold levels (linear regression):

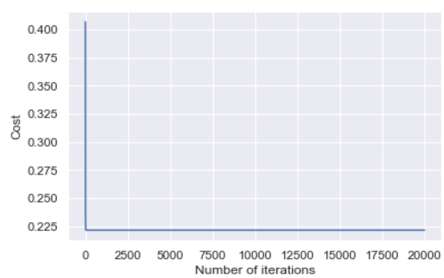
```
Threshold level 0.001

Cost Function converges at 14

Model Cost:  0.2264156137693888

Model Coefficients:
[[ 0.50203577  0.39637015  0.08442112 -0.39594155 -0.38179981  0.00059929
  -0.00129512 -0.01332445 -0.01542743 -0.04102379 -0.06070891 -0.00880489
  -0.0883344  -0.02479818]]
alpha: 0.2
```



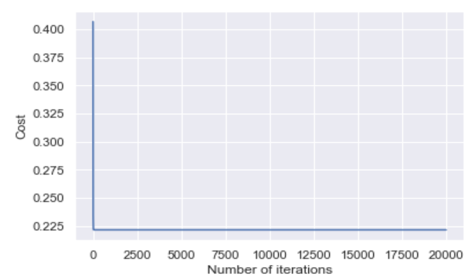

```
Threshold level 0.0001

Cost Function converges at 25

Model Cost:  0.22187601374255358

Model Coefficients:
[[ 0.50203577  0.39637015  0.08442112 -0.39594155 -0.38179981  0.00059929
  -0.00129512 -0.01332445 -0.01542743 -0.04102379 -0.06070891 -0.00880489
  -0.0883344  -0.02479818]]
alpha: 0.2
```



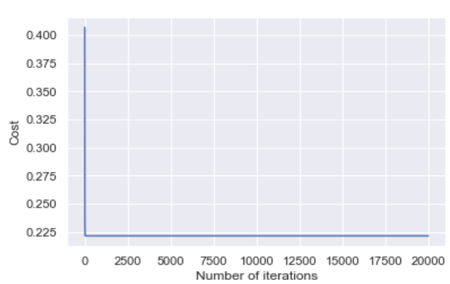

```
Threshold level 1e-05

Cost Function converges at 35

Model Cost:  0.22148187768679992

Model Coefficients:
[[ 0.50203577  0.39637015  0.08442112 -0.39594155 -0.38179981  0.00059929
  -0.00129512 -0.01332445 -0.01542743 -0.04102379 -0.06070891 -0.00880489
  -0.0883344  -0.02479818]]
alpha: 0.2
```

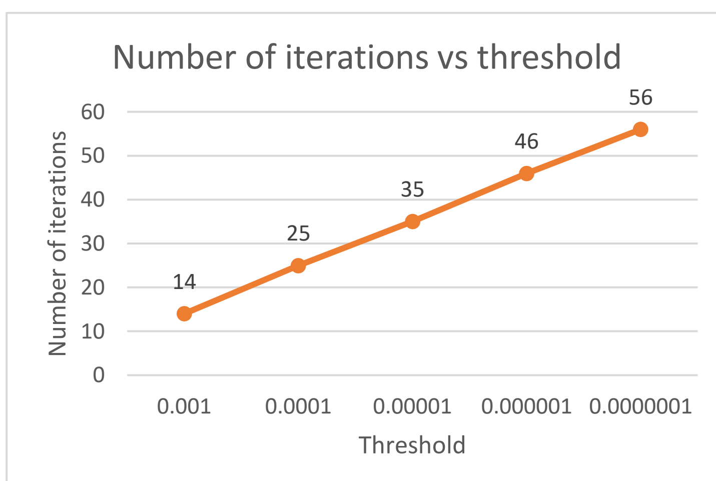


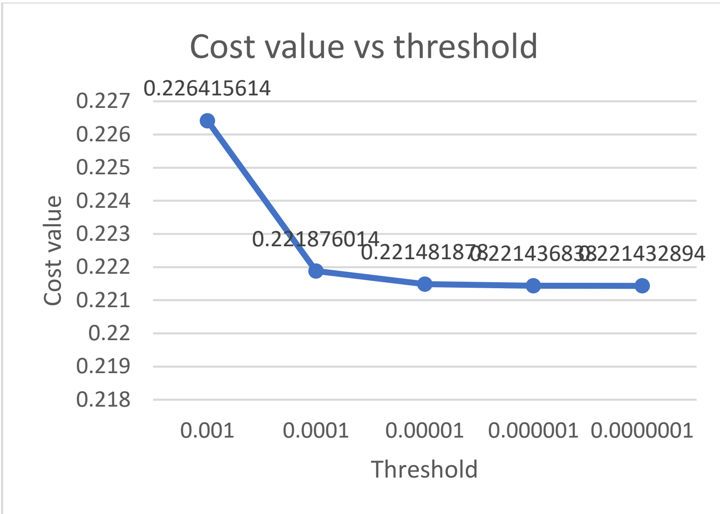

```
Threshold level 1e-06

Cost Function converges at 46

Model Cost:  0.2214368383330077

Model Coefficients:
[[ 0.50203577  0.39637015  0.08442112 -0.39594155 -0.38179981  0.00059929
  -0.00129512 -0.01332445 -0.01542743 -0.04102379 -0.06070891 -0.00880489
  -0.0883344  -0.02479818]]
alpha: 0.2
```

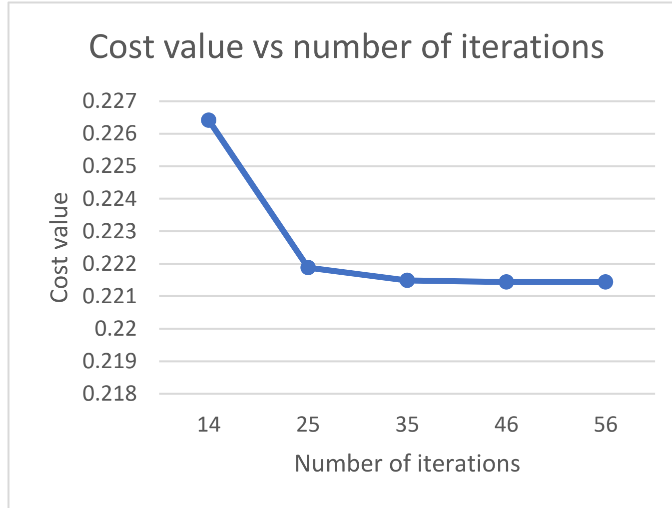



```
Threshold level 1e-07

Cost Function converges at 56

Model Cost:  0.2214328936484323

Model Coefficients:
[[ 0.50203577  0.39637015  0.08442112 -0.39594155 -0.38179981  0.00059929
  -0.00129512 -0.01332445 -0.01542743 -0.04102379 -0.06070891 -0.00880489
  -0.0883344  -0.02479818]]
alpha: 0.2
```









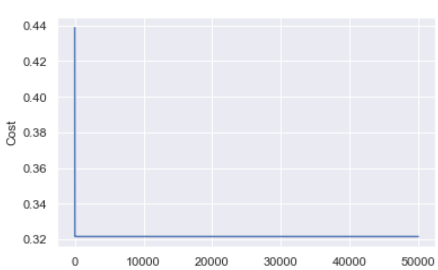
Threshold	Number of iterations	Cost value	Train MSE	Test MSE
0.001	14	0.226415614	0.44286479	0.44206195
0.0001	25	0.221876014	0.44286479	0.44206195
0.00001	35	0.221481878	0.44286479	0.44206195
0.000001	46	0.221436838	0.44286479	0.44206195
0.0000001	56	0.221432894	0.44286479	0.44206195

I have fixed the learning rate to 0.2 and varied learning rate as 0.001, 0.0001, 0.00001, 0.000001, 0.0000001. To find the best threshold value, I ran linear regression using in-built functions and matched the coefficients. Initial coefficients -0, initial alpha 0.5.

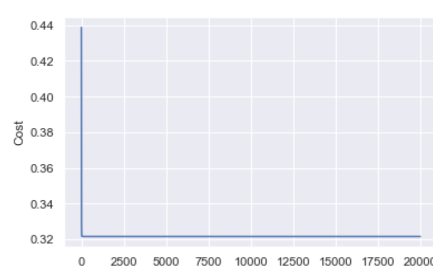
- For training data, threshold value "0.0000001" has the lowest cost function and converged at 56.
- Train MSE and test MSE are same for different threshold levels.
- As threshold decreases cost value also decreases.

Experiment 2 – Changing threshold levels (logistic regression):

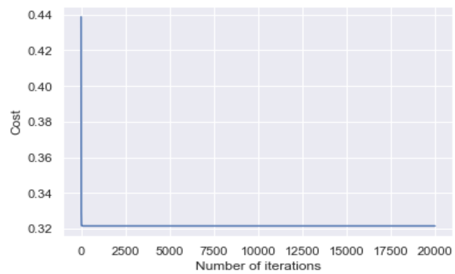
```
Threshold level 0.001

Cost Function converges at 12

Model Cost:  0.32605178376197813

Model Coefficients:
[[ 0.44917305  0.27504323  0.04886828 -0.28719797 -0.24562571 -0.01238468
  -0.02016434 -0.00444076 -0.02129589 -0.04532177 -0.10232155 -0.0107943
  -0.12658488 -0.02685845]]
alpha: 0.2
```



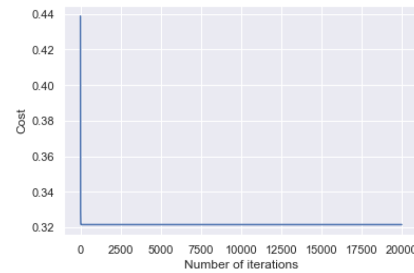

```
Threshold level 0.0001

Cost Function converges at 22

Model Cost:  0.32205971719240606

Model Coefficients:
[[ 0.44917305  0.27504323  0.04886828 -0.28719797 -0.24562571 -0.01238468
  -0.02016434 -0.00444076 -0.02129589 -0.04532177 -0.10232155 -0.0107943
  -0.12658488 -0.02685845]]
alpha: 0.2
```



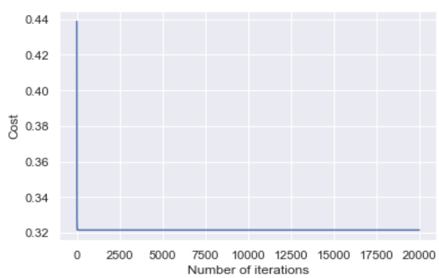
```
Threshold level 1e-05

Cost Function converges at 33

Model Cost:  0.32160779433052966

Model Coefficients:
[[ 0.44917305  0.27504323  0.04886828 -0.28719797 -0.24562571 -0.01238468
  -0.02016434 -0.00444076 -0.02129589 -0.04532177 -0.10232155 -0.0107943
  -0.12658488 -0.02685845]]
alpha: 0.2
```

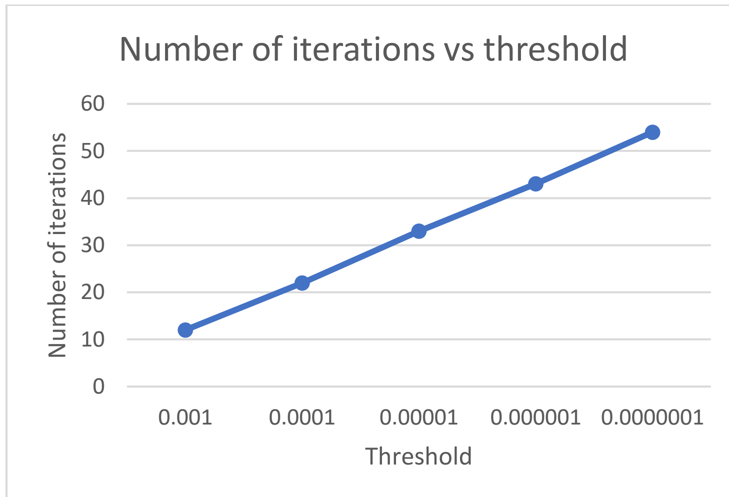


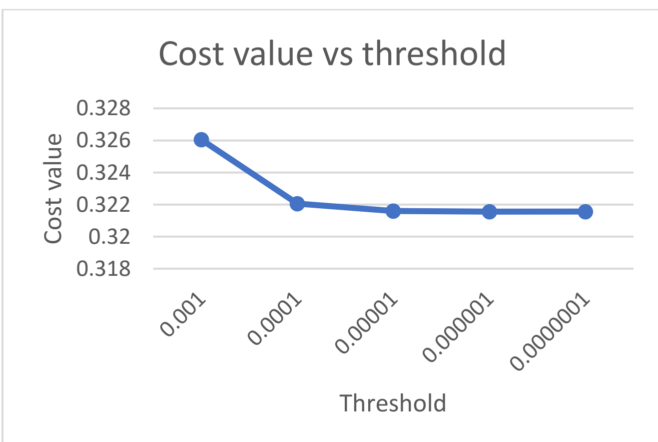

```
Threshold level 1e-06

Cost Function converges at 43

Model Cost:  0.32156792022212594

Model Coefficients:
[[ 0.44917305  0.27504323  0.04886828 -0.28719797 -0.24562571 -0.01238468
  -0.02016434 -0.00444076 -0.02129589 -0.04532177 -0.10232155 -0.0107943
  -0.12658488 -0.02685845]]
alpha: 0.2
```

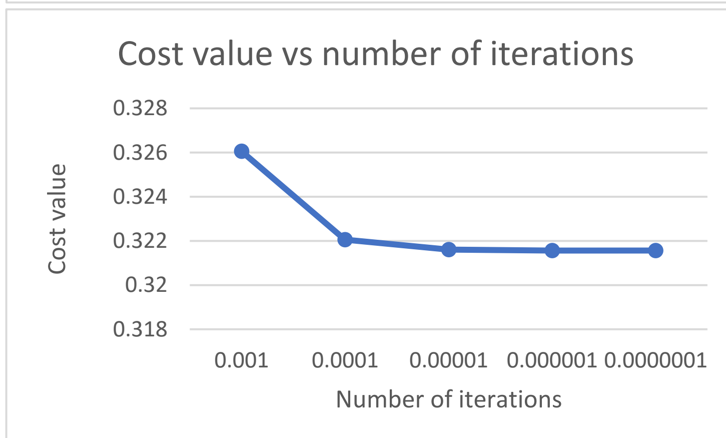



```
Threshold level 1e-07

Cost Function converges at 54

Model Cost:  0.32156330775823283

Model Coefficients:
[[ 0.44917305  0.27504323  0.04886828 -0.28719797 -0.24562571 -0.01238468
  -0.02016434 -0.00444076 -0.02129589 -0.04532177 -0.10232155 -0.0107943
  -0.12658488 -0.02685845]]
alpha: 0.2
```









Threshold	Number of iterations	Cost value	Train MSE	Test MSE
0.001	12	0.326051784	0.48205217	0.48105165
0.0001	22	0.322059717	0.48205217	0.48105165
0.00001	33	0.321607794	0.48205217	0.48105165
0.000001	43	0.32156792	0.48205217	0.48105165
0.0000001	54	0.321563308	0.48205217	0.48105165

I have fixed the learning rate to 0.2 and varied learning rate as 0.001, 0.0001, 0.00001, 0.000001, 0.0000001. To find the best threshold value, I ran logistic regression using in-built functions and matched the coefficients. Initial coefficients -0, initial alpha 0.5.

- For training data, threshold value "0.0000001" has the lowest cost function and converged at 54.
- Train MSE and test MSE are same for different threshold levels.
- As threshold decreases cost value also decreases.

Experiment 3 – Random variables:

Average_gpu_runtime = b0 + (b1 * MWG) + (b2 * NWG) + (b3 * KWG) + (b4 * MDIMC) + (b5 * NDIMC) + (b6 * MDIMA) + (b7 * NDIMB) + (b8 * KWI) + (b9 * VWM) + (b10 * VWN)

	Alpha - 0.2			**Threshold - 0.0000001**		
	Cost value	**Converging point**	**Train MSE**	**Test MSE**	**Train R-square**	**Test R-square**
Model - 10 random variables	0.22752109	56	0.45369907	0.45362344	0.547533818	0.547533818
Best model from experiment 1 and 2	0.22143289	56	0.44286479	0.44206195	0.557135208	0.559139317

- Model with 10 random variables perform poorly. As its cost value, train MSE and test MSE are higher than the best model of 16 variables.
- Train R-square and test R-square are lower than the best model in experiment 1 and 2.

Experiment 4 – Selected variables:

Average_gpu_runtime = b0 + (b1 * KWG) + (b2 * MDIMC) + (b3 * NDIMC) + (b4 * MDIMA) + (b5 * NDIMB) + (b6 * KWI) + (b7 * VWM) + (b8 * VWN) + (b9 * SA) + (b10 * SB)

	Alpha - 0.2			**Threshold - 0.0000001**		
	Cost value	**Converging point**	**Train MSE**	**Test MSE**	**Train R-square**	**Test R-square**
Model - 10 best variables	0.22752109	56	0.45369907	0.45362344	0.547533818	0.547533818
Best model from experiment 1 and 2	0.22143289	56	0.44286479	0.44206195	0.557135208	0.559139317
Model – 10 random	0.43415258	33	0.86830447	0.8714047	0.131695526	0.130963273
Model all variables	0.22143289	56	0.44286479	0.44222978	0.557135208	0.55897195

Logistic Regression results

	Learning rate : 1.25				
	Train accuracy	**Test Accuracy**	**Precision**	**Recall**	**F1-square**
Model - 10 best variables	0.869	0.87	0.87	0.87	0.87
Best model from experiment 1 and 2	0.885	0.884	0.88	0.88	0.88
Model - 10 random variables	0.655	0.654	0.77	0.78	0.77

Model all variables	0.885	0.884	0.88	0.88	0.88

Interpretation of the results:

- The model with 10 best variables performed better than the other two models (Model – 10 random variables and Best model from experiment 1 and 2).
- It has a lower cost value, train MSE and test MSE.
- It has a better train R-square and test R-square values.
- Model with 10 best variables gave the better accuracy, precision, recall, f1-square than the model – 10 random variable.
- But the model with all the variables out performed all the other models in terms of cost value, train MSE, test MSE, train and test r-square, accuracy, recall, precision, f1-square.
- Selecting features by my own choice did not perform better than using all the features.

What do you think matters the most for predicting the GPU run time?

- The top important feature are MWG, NWG, VWM, VWN which help predict the GPU run time.

What other steps you could have taken with regards to modeling to get better results?

- We can use hyperparameter tuning deploy algorithm, support vector regressor, decision tree regressor, gradient boosting, neural network etc. to get lower MSE values.
- We can use dimension reduction techniques that uses an orthogonal transformation to convert a set of possibly correlated variables into a set of values of linearly uncorrelated variables.