BUAN 6341 Applied Machine Learning

ASSIGNMENT 1

SGEMM GPU Kernel performance prediction and classification

Introduction:

The objective of this project is to predict GPU run time and convert the same problem into binary classification. First task is to implement a linear regression using gradient descent to predict the GPU runtime. Secondly the dataset needs to be converted to binary classification and implement logistic regression using gradient descent. This report details the various experiments conducted using the dataset to understand the affect of the gradient descent algorithm. It also, shows the effect of the use of various independent variables/features in selecting the best model.

About the Data:

The data set is of SGEMM GPU kernel performance which consists of 14 features and 241600 records. 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 261400 possible parameter combinations. Out of 14 features, the first 10 are ordinal and can only take up to 4 different powers of two values, and the 4 last variables are binary.

Project Outline:

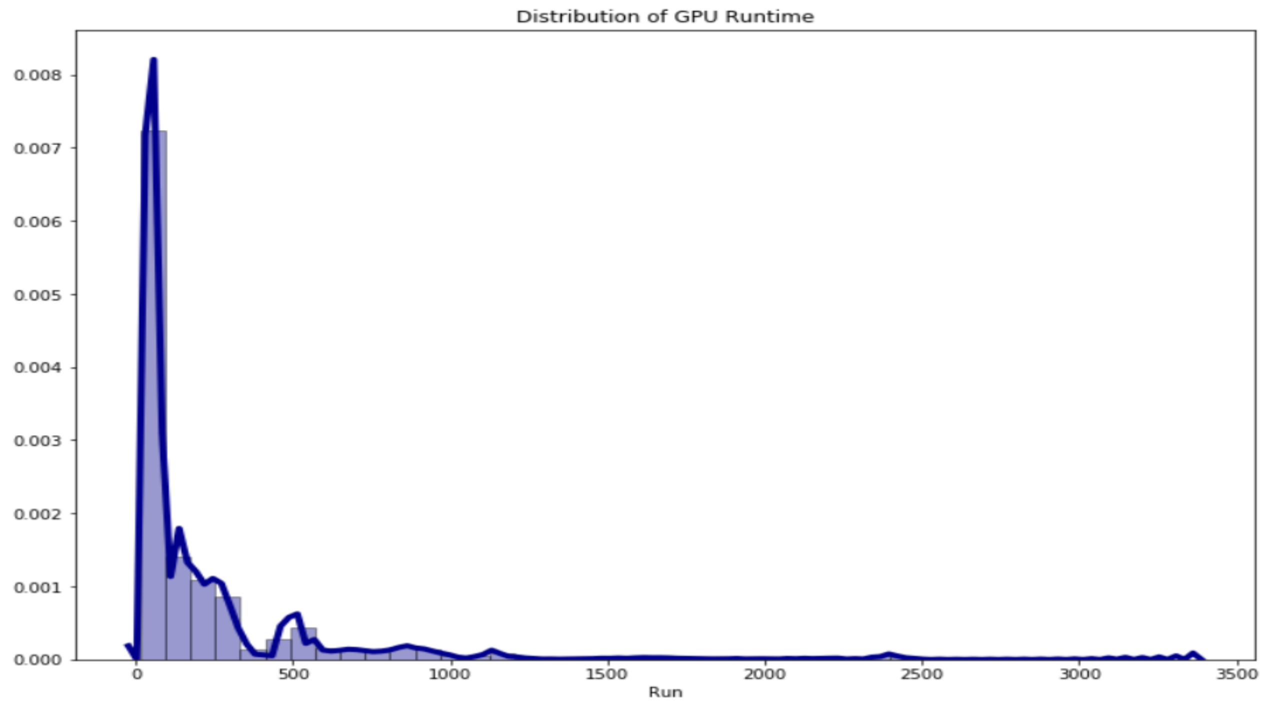
Algorithm Implementation

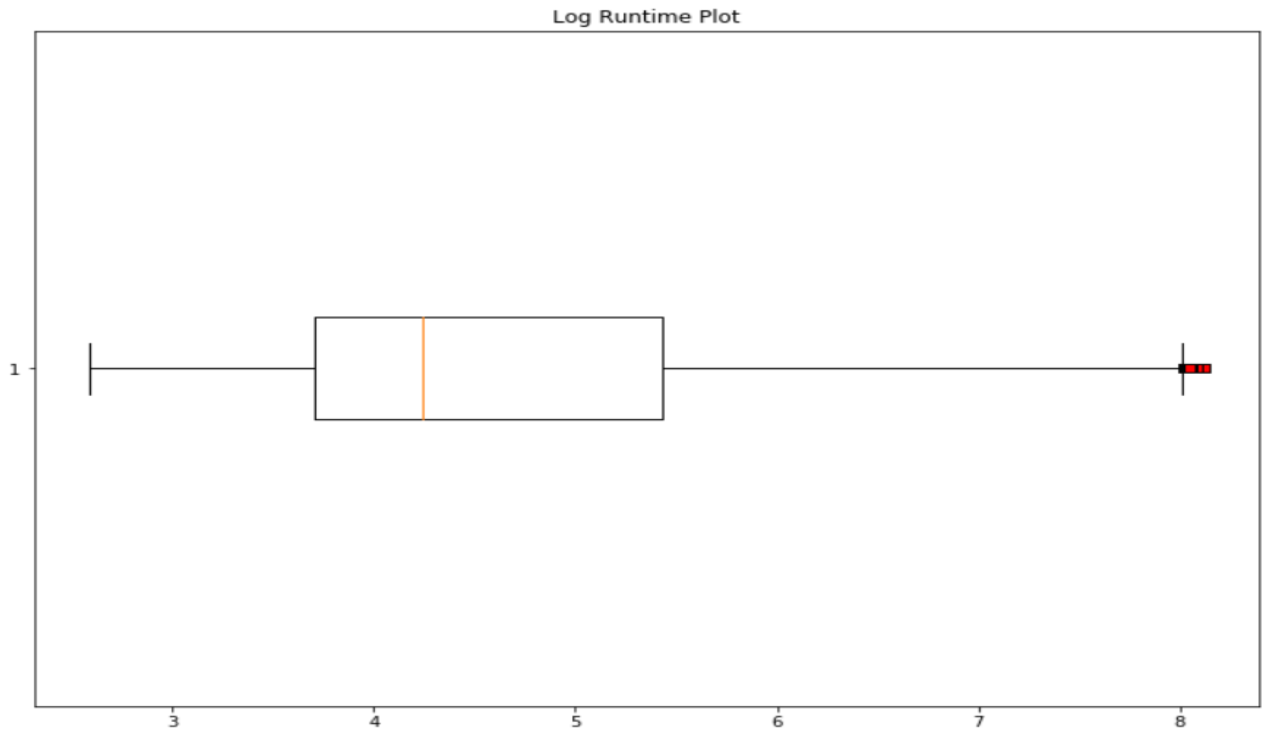
The gradient descent algorithm is implemented using python numerical computation package “numPy”. The numPy package provides new homogenous array and matrix data structures to python which is immensely beneficial to implement vectorized implementation of the gradient descent.

The algorithm is implemented for two different machine learning techniques, namely, Linear regression and Logistic regression. The algorithm is implemented with options to change the hyperparameters: Learning rate(alpha), convergence threshold, and max epoch.

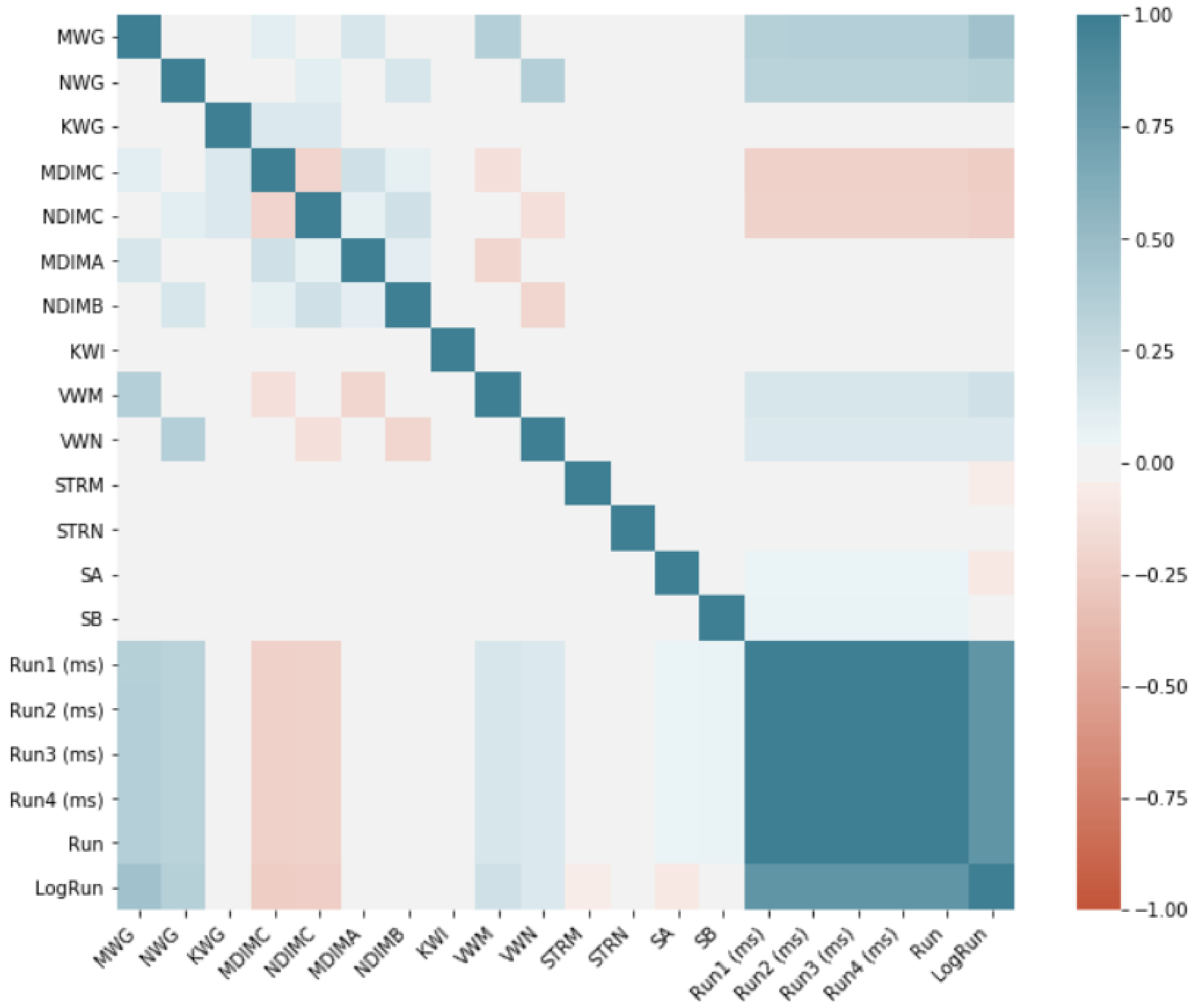
Data Preparation and Exploratory Data Analysis

The objective of the Linear regression model is to accurately predict the GPU runtime. From the figure you can see that, the distribution of the average GPU Runtime is positively skewed because of the outliers, so converting the value to log scale helps in better modeling as you can see the boxplot.


Distribution of GPU Runtime
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
500
1000
1500
2000
2500
3000
3500
Run


Log Runtime Plot
1
3
4
5
6
7
8

The correlation plot doesn't indicate nay high correlation between the parameters. However, we can see that there is a strong relationship between dependent variable "LogRun" and independent variables "MWG", "NWG", "MDIMC", "NDIMC", "VWM" "VWN". So, including these variables in the model increases the power to detect the effect of other predictors.



Scaling wasn't performed on the features since out of 14 features, 10 are ordinal and 4 are categorical. The result might not be greatly impacted when ordinal variables are not scaled.

For **Logistic Regression**, the regression problem was converted into a classification problem by classifying all values above the mean runtime as 1 and value below as 0 . By this classification, we are trying to predict when the GPU runtime is high.

Experimentation and Results

Mainly 4 experimentation was undertaken using the dataset in which we explored the effect of changing the hyperparameters of the algorithm and discuss the effectiveness of feature selection in predictive accuracy. These experiments were repeated both for linear and logistic regression. Finally, the results of the experiments were discussed in the results section and further improvement opportunities were discussed.

Experimentation

1. Hyperparameter Tuning

Linear Regression was implemented using gradient descent algorithm, the hyperparameter we can tune is learning rate (∝). The value of learning rate must be tuned for optimal convergence to estimates (β values).

$$\begin{aligned}\log(\hat{y}) = &\beta_0 + \beta_1 \cdot MWG + \beta_2 \cdot NWG + \beta_3 \cdot KWG \\ &+ \beta_4 \cdot MDIMC + \beta_5 \cdot NDIMC + \beta \cdot MDLMA \\ &+ \beta_7 \cdot NDIMB + B_8 \cdot KWI + \beta_9 \cdot VWM \\ &+ \beta_{10} \cdot VWN + \beta_{11} \cdot STRM + \beta_{12} \cdot STRN \\ &+ \beta_{13} \cdot SA + \beta_{14} \cdot SB\end{aligned}$$

Cost Function:

$$J(\beta) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2$$

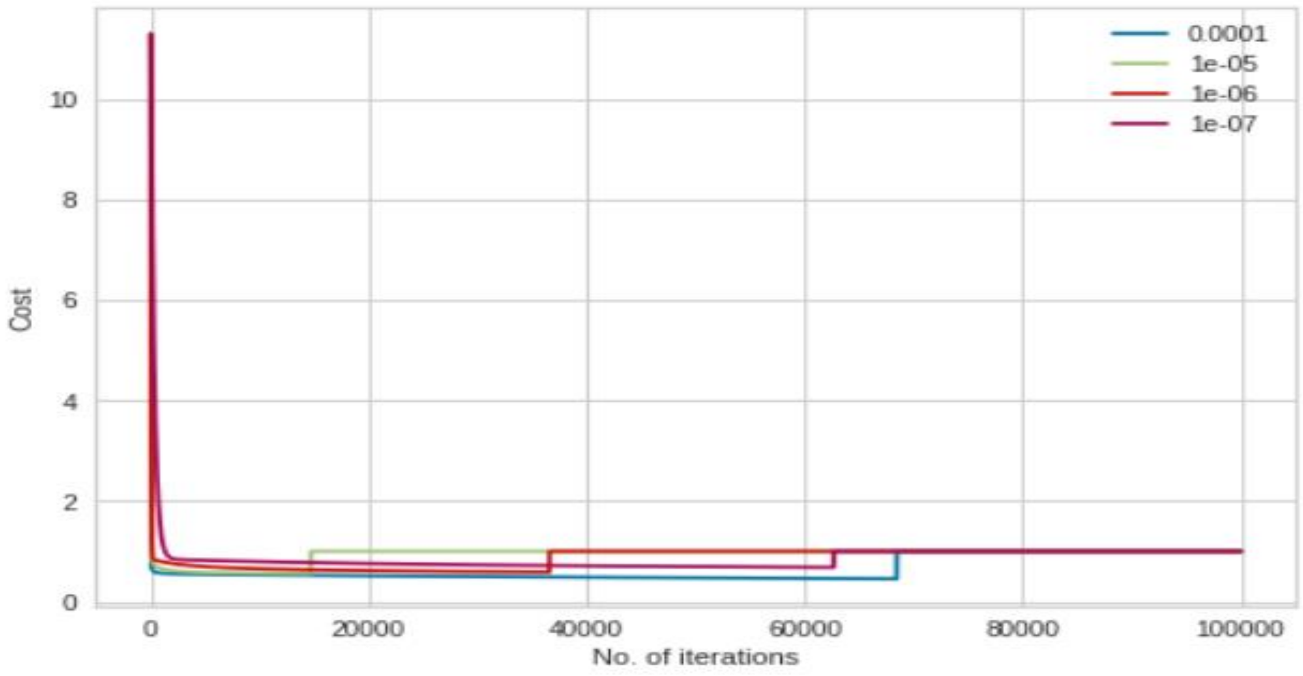
Gradient Descent:

$$\beta_j := \beta_j - \alpha \frac{1}{m} \sum_{i=1}^{n} \left(\hat{y}^{i} - y^{i} \right) x_j^{i}$$

With each epoch, the cost is seen continuously decreasing and the algorithm is said to have reached convergence when the decrease in cost is within a defined threshold. When the learning rate is low, the algorithm takes many iterations to converge while the estimates can be more precise. When learning rate is very high, the algorithm might diverge and never reach the minimal point.

The figure shows train\test Cost as a function of learning rate. As we can observe, when learning rate is very low (∝ = 0.00001), the cost is higher. This is because of slow learning, even at the 10000 iterations the algorithm did not converge with a tolerance value 0.000001. The effect of changing tolerance value can be seen in a later experiment.

Alpha	0.0001	0.00001	0.000001	0.0000001
Iterations(Epochs)	100000	100000	100000	100000
Steps to Converge	63519	14685	37033	62486
Training Cost	0.4218	0.5246	0.5570	0.6484
Test Cost	0.4582	0.5565	0.5861	0.6812



The **logistic regression** was implemented using gradient descent algorithm, the hyperparameter we can tune is learning rate (∝). The intuition of changing the learning rate is like that of linear regression. By picking a good learning rate, the algorithm converges to the global minima faster than stuck at a saddle point.

Hypothesis:

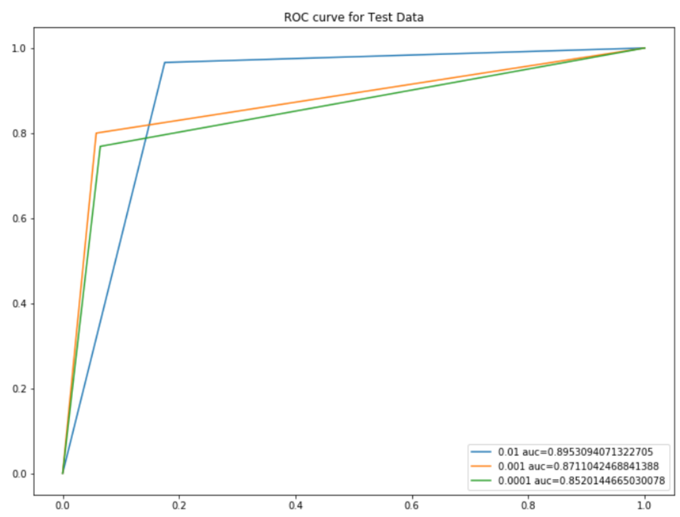
$$\hat{y}i = \frac{1}{1 + e^{-\beta^\top x^i}}$$

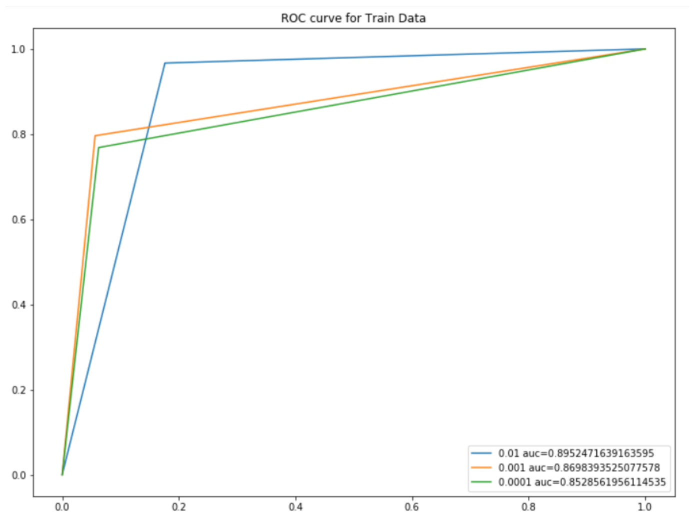
Cost Function:

$$J(\beta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^i \log\left(\hat{y}^i\right) + (1 - y^i)\log\left(1 - \hat{y}^i\right)\right)$$

For the threshold of 0.5, and $\propto = 0.001$ the accuracy of training & test were approximately around 90%

Alpha	0.01	0.001	0.0001
Iterations(Epochs)	100000	100000	100000
Steps to Converge	61821	11351	11554
Training Accuracy	0.86125	0.90491	0.89309
Test Accuracy	0.86124	0.90546	0.89220



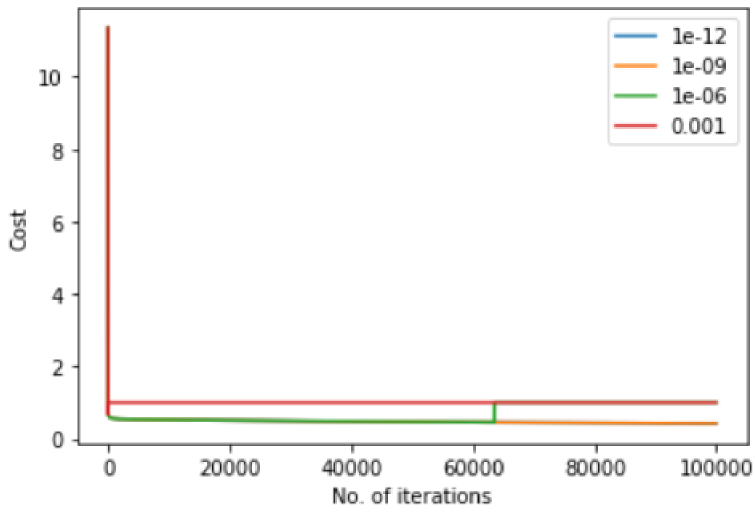


2. Changing Convergence Threshold

When the change in Cost is within the Convergence threshold, the gradient descent algorithm is said to be converged. So, the algorithm converges faster at higher threshold value but at a certain point increasing threshold is found to increase the cost in both train and test dataset. We find the optimal value of threshold by plotting cost as a function of convergence threshold and find the point at which the test error is minimum.

Linear regression

Threshold	10^-12	10^-9	10^-6	10^-3
Steps to converge	Didn't converge	Didn't converge	63519	63
Test Cost	0.421858	0.421858	0.45436	0.67581
Training Cost	0.42504	0.42504	0.45827	0.680376

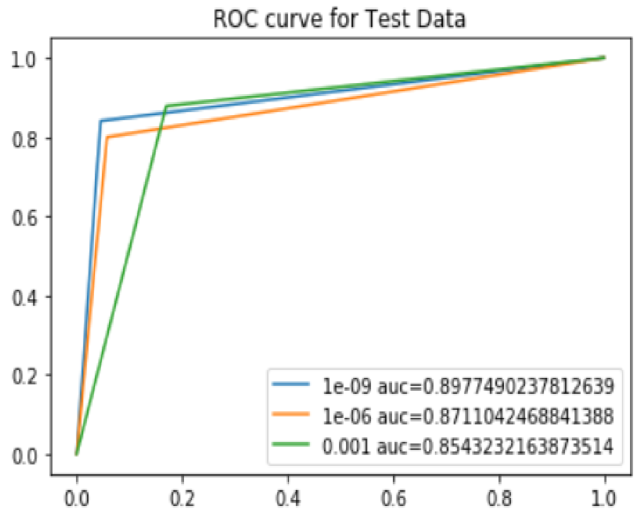


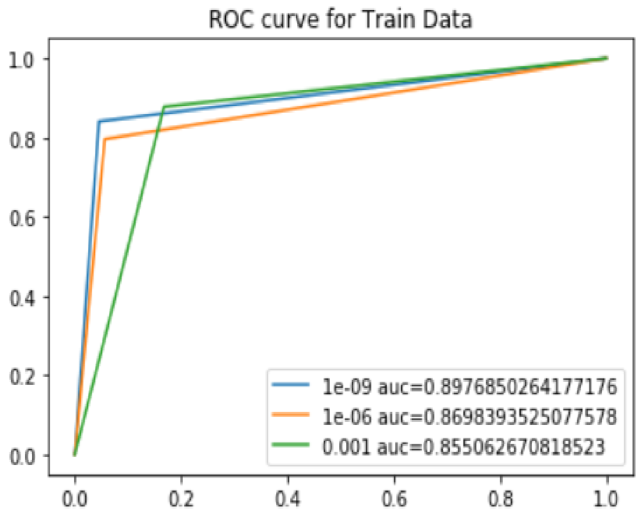
Interesting observation is that no of iterations is found to nearly constant after a certain threshold (~10^-6 for this setting). This is because the algorithm converged to global

Logistic Regression

The best Threshold for this dataset is 10^-9

Threshold	10^-9	10^-6	10^-3
Steps to converge	Didn't converge	11351	172
Test Accuracy	0.92516	0.9054	0.8426
Training Accuracy	0.9246	0.9049	0.84416





3. Random Feature Selection

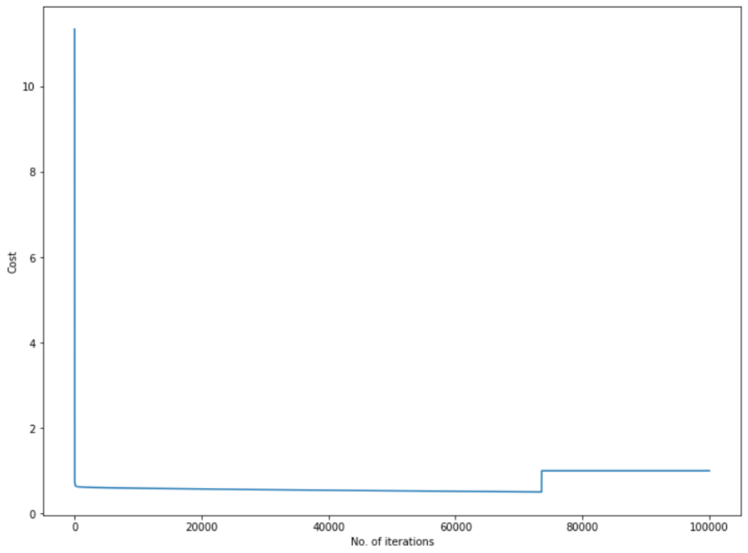
In this experiment, 8 features are selected at random using the DataFrame.sample function and the train and test errors are compared to the errors from the original set of features. The errors of random feature selection are found to be more than the respective errors from the original set of 14 features.

$$\log(\hat{y}) = \beta_0 + \beta_1 \cdot MW_G + \beta_2 \cdot VWN + \beta_3 \cdot STRN + \beta_4 \cdot VWM + \beta_5 \cdot MDIMA + \beta_6 \cdot KWI + \beta_7 \cdot SB + \beta_8 \cdot KWG$$

Linear Regression

The model was run keeping $\propto = 0.001$, threshold$=0.000001$ & epochs$=100000$. Here it took 73560 steps to converge and the error found to increase by 19.67%

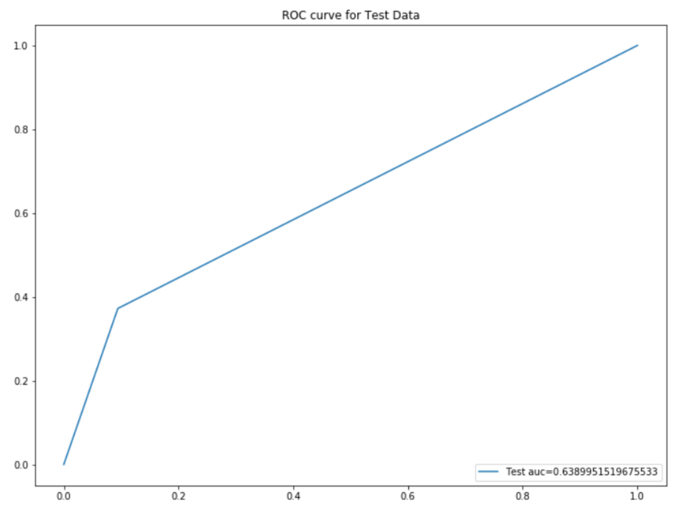
Data	Original Features	Random Features	% Change
Train	0.4218	0.50477	19.67
Test	0.4582	0.62779	37.01

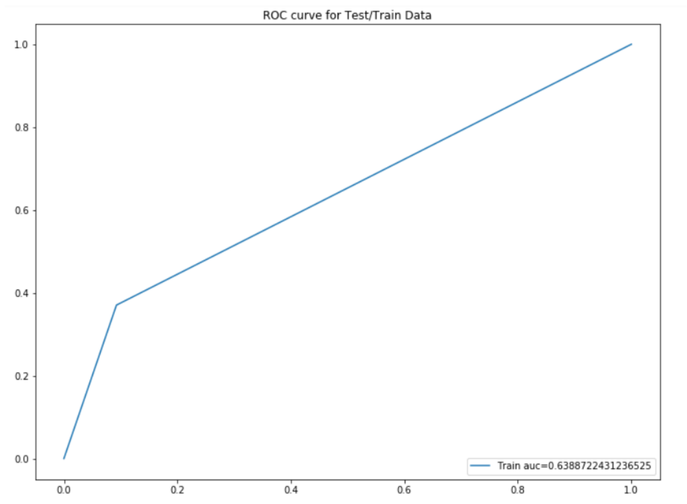


Logistic Regression

Logistic Regression was performed on 8 randomly selected features with $\propto = 0.001$, threshold$=0.000001$ & epochs$=100000$. The train & train accuracy reduced by 15.17% & 15.22% respectively.

Data	Original Features	Random Features	% Change
Train	0.90491	0.7676	15.17
Test	0.90546	0.7676	15.22





4. Manual Feature Selection

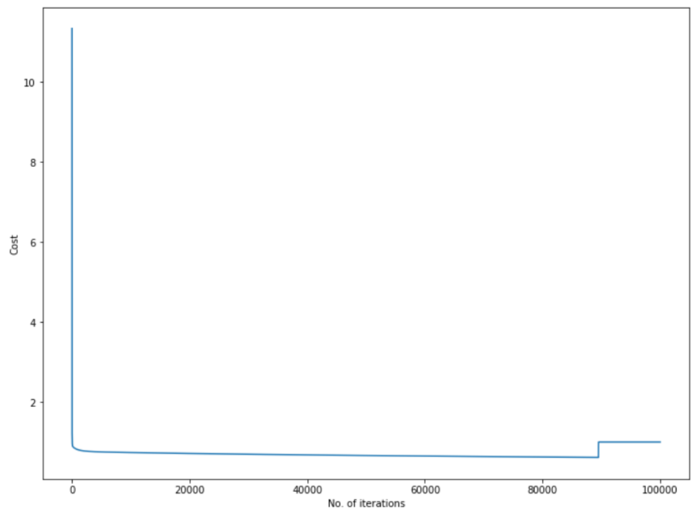
In this experiment 8 features are handpicked based on the correlation matrix

$$\begin{aligned}\log(\hat{y}) = {} & \beta_0 + \beta_1 \cdot MWG + \beta \cdot VWN + \\ & \beta_3 \cdot STRN + \beta_4 \cdot VWM + \beta_5 \cdot MDIMA \\ & + \beta \cdot KWI + \beta_7 \cdot SB + \beta_8 \cdot KWG\end{aligned}$$

Linear Regression

The model was run keeping ∝=0.001, threshold=0.000001 & epochs=100000. Here it took 89515 steps to converge and the error found to increase by 45.86%

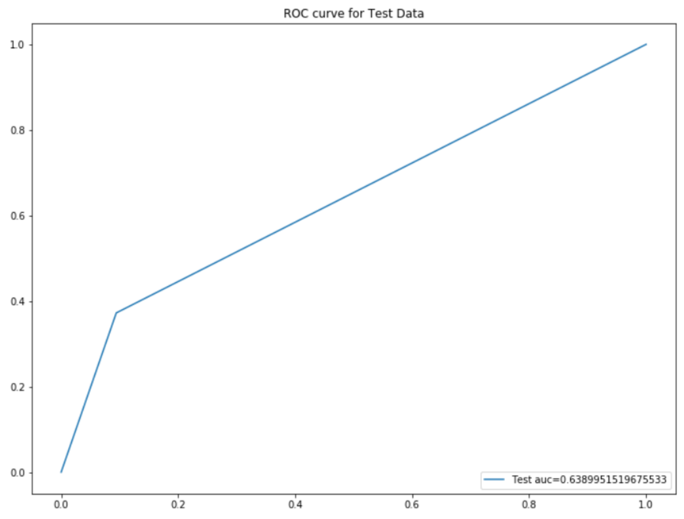
Data	Original Features	Handpicked Features	% Change
Train	0.4218	0.61527	45.86
Test	0.4582	0.62779	36.94

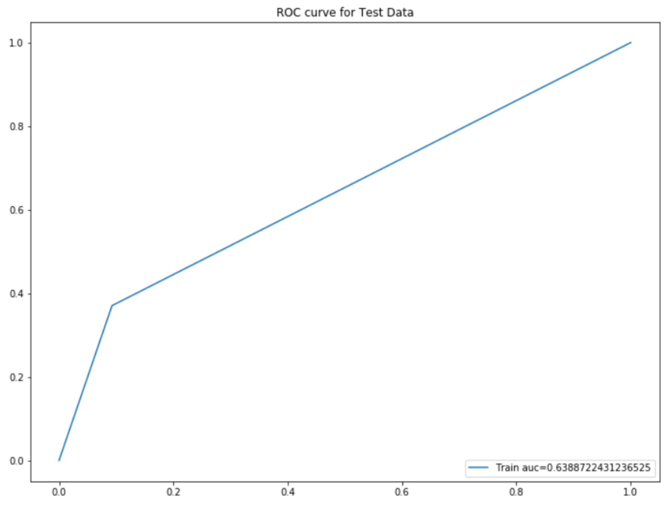


Logistic Regression

Logistic Regression was performed on 8 handpicked features with ∝=0.001, threshold=0.000001 & epochs=100000. The train & train accuracy reduced by 15.29% & 15.22% respectively.

Data	Original Features	Random Features	% Change
Train	0.90491	0.7665	15.29
Test	0.90546	0.7676	15.22





Results

Through experimentation, we found that optimizing hyperparameters like learning rate and convergence threshold are effective for better accuracy of the prediction. But some accuracy improving measures like low alpha settings or very low convergence threshold increases the time taken for fitting the ML algorithm with the training set. Therefore, machine learning involves a balancing between accuracy and computational limitations.

Also, we saw the importance of feature selection and engineering to improve the performance of the ML algorithm. Restricting features and randomly selecting features without applying domain knowledge performs worse than a carefully executed ML model. Such models incorporate some of user's domain knowledge which improves the acquired knowledge from data.

Going forward , we can improve the algorithm by adaptively changing the learning rate. Such an algorithm can give higher learning rate at the start and progressively decrease the learning after each epoch. Such an approach can greatly reduce the number of iterations required by the algorithm to converge. Also need to use domain knowledge to understand which features and functional form are important for prediction is key for good learning of the algorithm. To further improve the performance, feature scaling can be used before training the model.