Group Two

Dr. Tweneboah

CMPS 620

11 May 2021

Maximizing GPU Performance and Efficiency in Distributed Computing Systems

Group Two Members:

- Kyle Calabro
- Mikhail Delyusto
- Joana Dervishaj
- Miguel Esteban Diaz

Executive Summary:

This dataset, found on the UCI Machine Learning Repository contains running times for a matrix-matrix product A * B = C, where all matrices are of size 2048 by 2048. The data set is composed of 241,600 observations. Each observation represents one possible combination of parameters for the Single Precision General Matrix Multiplication (SGEMM) GPU Kernel that was utilized during this experiment. The dataset is composed of eighteen attributes, fourteen of such attributes represent the various parameters used by the SGEMM GPU Kernel, the remaining four attributes are the runtimes of four test runs for the given computation with the given parameters in milliseconds (ms) for that particular configuration of parameters.

Our primary objective with this project is to produce a Machine Learning model that is capable of accurately predicting if the runtime of a given computation with the provided parameters of the SGEMM GPU Kernel will be “slow” or “fast”. Within this context, our intention is to convert this to a binary classification problem, utilizing supervised learning methods to predict the labels. In this particular case, we will average the four given runtimes for

each observation and then use the mean of the averaged value to determine what is to be deemed a “slow” and “fast” runtime; “slow” runtimes are greater than the mean whereas “fast” runtimes are less than the mean runtime.

It is our belief that such a model could be deployed within a distributed computing environment to maximize the performance and efficiency of such an environment. Such an environment may potentially be composed of a variety of different machines with varying specifications. Some machines are more likely to have higher specifications and in turn higher performance than others. In this case, our proposed model could be used to route certain processes to certain machines in order to maximize the environment's use of its available resources. Within this context, our model could also perhaps be expanded into a multiclass classification model and offer greater efficiency for a given environment or even be incorporated in an online learning system. This is of course highly dependent on the composition of the given environment and the actual computations being performed as well as the machines performing them. Therefore, for the sake of simplicity and as a proof of concept we chose to keep our model(s) limited to binary classification.

To accomplish this objective we conducted a fair amount of exploratory data analysis and preprocessing to further our understanding of the dataset and ensure the quality of the data was satisfactory for our needs. We then utilized a training, validation and testing set with a seventy, fifteen and fifteen percent split(s), respectively, to train and evaluate the performance of the various models we implemented (Appendix 1.M). From evaluation of the models on the validation set, we found that a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel (Appendix 1.P), a Decision Tree using Gini as the impurity measure (Appendix

1.R) and a Neural Network (Appendix 3.A) offered the best performance(s) with regards to accuracy, precision and recall.

Deploying the previously mentioned models on the test set yielded impressive results. The SVM with an RBF kernel yielded accuracy, precision and recall scores of 0.94, .96 and .89, respectively (Appendix R.A). Our Decision Tree using Gini as the impurity measure yielded accuracy, precision and recall scores of .98, .97 and .97, respectively (Appendix R.B). The Neural Network offered similar performance with accuracy, precision and recall scores of .97, .97 and .96, respectively (Appendix R.C). Overall, the performance of these three models is rather exceptional and in turn shows us that a model such as the one we are proposing, when deployed in a distributed computing environment to route certain processes to particular machines has the potential to offer a large advantage in efficiency and performance.

Whilst it did not perform as well as the other models above, the Random Forest model we implemented did offer some important insight into what features have the greatest effect on the runtimes for these computations in the way of the feature importance plot such models can provide. From the feature importance plot (Appendix 4.A) we observed that the features "NWG" and "MWG" had the greatest impact on our predictions. This was confirmed by the correlation matrix as well (Appendix 1.J) These particular features represent the per-matrix 2D tiling value(s). Conducting some further research into tiling within this context of matrix multiplication, we found that "global memory accesses with tiling is half the global memory accesses" without tiling (Xu, Penny). This in turn implies that with "tiling value NxN, potential memory reduction of global memory traffic would be N" (Xu, Penny). As such, we concluded that the features "NWG" and "MWG" had the most profound impact on the runtime of a given computation of all features within the dataset.

Main Report:

Modern GPUs are progressing at an incredibly rapid pace with each generation still making great performance improvements over the previous iterations. With the use of GPUs becoming ever more prominent in the world of parallel computing, artificial intelligence and even cluster computing, a model such as the one we are proposing could be of great use in the real world. In the ethos of mass computing and data processing this could heavily optimize a company's use of processor time and how to best make use of all its available resources. This is especially important in our modern, data-driven world in which we are now processing more data than ever before.

It is important to note that this particular dataset was formed from an experiment performed on a machine utilizing an Intel Core i5 processor with sixteen gigabytes of RAM and a Nvidia Geforce GTX 680 with four gigabytes of VRAM. By today's standards this is a relatively underwhelming machine in terms of specs for heavy computing tasks but nonetheless the data provided by the experiment will serve our needs within this context.

In order to achieve our objective we were required to conduct exploratory data analysis (EDA), perform preprocessing on the data, deploy and evaluate a number of models on validation and test sets, as well as interpret the results. To identify any potential issues with overfitting five-fold cross-validation was also utilized for many of the models we explored. With the intent that our final model could be deployed within a distributed computing environment to route certain processes to certain machines and potentially be utilized in an online learning system, we were not only concerned with the performance of our models with regards to metrics such as accuracy, precision and recall but also the training time(s) of the models themselves.

Given our chosen dataset and objective of predicting the runtimes of matrix multiplication computations categorically with labels of “fast” and “slow” we were able to explore a variety of models to find what best suited this supervised binary classification problem. Such models include Decision Trees, Support Vector Machines (SVM), Boosting, Bagging, Random Forests, Logistic Regression, Neural Networks, Stochastic Gradient Descent (SGD) and Pasting.

Beginning the project, we first converted the four given runtimes for each observation to a single value representing the average of the provided runtimes (Appendix 1.A). We then of course checked for missing values within the dataset, of which we found none (Appendix 1.B). To gain a better understanding of the distribution of the newly averaged runtimes, we utilized a boxplot for the runtime feature. The boxplot revealed that there were a number of outliers a slightly beyond the 500 millisecond mark for the runtime feature (Appendix 1.C). We decided that it would be best to exclude these from the dataset as such outliers could be caused by the process running the computation on the GPU being interrupted by input/output or a variety of other factors. To resolve the outliers, we filtered out the runtime values below the lower hinge and above the upper hinge represented from the box plot (Appendix 1.D). To confirm the outliers were resolved an empirical cumulative distribution plot was used (Appendix 1.E). With the outliers resolved in our target variable to-be, we were able to move forward and examine other features.

Histograms were utilized to visualize the distributions for all attributes of the dataset (Appendix 1.F). The distribution(s) for many of the attributes were fairly balanced as the dataset contains observations for all possible combinations of the parameters utilized by the SGEMM GPU kernel. Notably however, the runtime appeared to be right-skewed (Appendix 1.G). To correct this issue we determined that transforming the runtime by applying the natural-log to each of its values would be the best solution (Appendix 1.H). Another histogram was then utilized to confirm the transformation and ensure the issue had been resolved (Appendix 1.I). At this point we were confident that any issues with the quality of the data had been identified and resolved.

A correlation matrix was then utilized to further our understanding of how the features were correlated amongst themselves, if at all (Appendix 1.J). From the correlation matrix we observed that “NWG” and “MWG” had the highest positive correlation with the runtime feature at a value of .37 and .26, respectively. A few features did exhibit some negative correlation amongst each other however, we determined these would not require further investigation though as their correlation values were not extremely concerning with a minimum of -.23. With the information gathered from the correlation matrix and previously mentioned EDA we were satisfied that we had gained a fundamental understanding of the data we were working with and that it was of satisfactory quality for our needs in achieving our objective.

Our objective required us to convert this to a binary classification problem. As such, we utilized the mean of the runtimes as the threshold for our cutoff point (Appendix 1.K). Any runtime above the mean would be converted to a 1 to represent a “slow” runtime. Conversely, any runtime below the mean would be converted to a 0 to represent a “fast” runtime. We used integers for the labels in this case as it would aid in speeding up our computations ever so slightly given the original size of the dataset at 241,600 observations. After converting runtimes to a binary value we used another histogram to visualize the distribution of classes to ensure they were balanced within an acceptable tolerance (Appendix 1.L).

To best evaluate and tune our models we utilized training, validation and test sets. The training set was composed of seventy percent of the remaining original dataset whilst the validation and test sets were composed of the remaining thirty percent and split evenly into fifteen percent each (Appendix 1.M). The features were then also standardized via scikit-learn's StandardScaler package. This concluded our exploratory data analysis and preprocessing of the data, it was now time to implement and evaluate models on the validation set to discover the ones best suited to our needs.

To begin working with models and evaluating their performance on the validation set, we first implemented Support Vector Machines (SVM) with linear, polynomial, and Radial Basis Function (RBF) Kernels. The SVM model utilizing a linear kernel exhibited accuracy, precision and recall scores of .85, .84 and .77, respectively (Appendix 1.N). Utilizing a polynomial kernel offered slightly improved performance with accuracy, precision and recall scores of .89, .91 and .81, respectively (1.O). Switching to an RBF kernel offered the greatest performance out of all the SVM models with accuracy, precision and recall scores of .94, .96 and .89, respectively (Appendix 1.P). The SVM models were however particularly slow to train in comparison to the other models we would go on to implement.

Decision Trees were also implemented utilizing both Entropy and Gini as impurity measures. Our Decision Tree with Entropy as the impurity measure offered accuracy, precision and recall scores of .98, .97 and .97, respectively (Appendix 1.Q). With Gini as the impurity measure the Decision Tree offered marginally better scores of .98, .98 and .978 for accuracy, precision and recall, respectively (Appendix 1.R). Attempting to prune the Decision Tree utilizing Gini as the impurity measure with GridSearch yielded slightly lower performance with an accuracy of .976 (Appendix 1.S). While both Decision Tree models exhibited exceptional performance overall, the model utilizing Gini as the impurity measure was slightly faster to train than the model with Entropy as the impurity measure.

Two boosting models were then implemented and evaluated on the validation set. These models utilized Gradient Boosting and AdaBoost. The Gradient Boosting model offered mediocre performance with accuracy, precision and recall scores of .75, .8 and .47, respectively (Appendix 1.T). The AdaBoost model yielded better performance with scores of .8, .76 and .7 for accuracy, precision and recall, respectively (Appendix 1.U). Compared to the models we had implemented thus far, the boosting models had underperformed when evaluated on the validation set and would not be considered for deployment on the test set.

A Stochastic Gradient Descent (SGD) model was then implemented. The SGD model offered similar performance to the boosting models with accuracy, precision and recall scores of .8, .75 and .73, respectively (Appendix 2.A). We also explored utilizing Mini-Batch Gradient Descent within a Neural Network. With a batch size of 32 and 30 epochs the Neural Network model offered impressive performance with scores of .96, .97 and .93 for accuracy, precision and recall, respectively (Appendix 2.B). This particular Neural Network was composed of four total layers with the hidden layers utilizing the Rectified Linear Unit (ReLU) activation function, with the output layer utilizing the Sigmoid activation function as this is a binary classification problem.

Another Neural Network model was also implemented, utilizing the ReLU activation function for the inner layers and the Adam optimization function. This particular model also offered exceptional performance despite the inner layers only consisting of twenty, fifteen, and ten neurons each and being trained with fifteen epochs. With scores of .98, .98 and .96 for accuracy, precision and recall, respectively, this was one of our top performing models when evaluated on the validation set (Appendix 3.A).

A Pasting model with five hundred Decision Trees was also implemented with one hundred as the max number of samples. This particular model offered mediocre performance with accuracy, precision and recall scores of .85, .88 and .71, respectively (Appendix 3.B). A Bagging model was also utilized with the same hyperparameters aside from the bootstrap flag. This particular model, interestingly, offered the exact same performance as the Pasting model with scores of .85, .88 and .71 for accuracy, precision and recall (Appendix 4.B). These models would not be considered for deployment on the test set given their performance.

We also explored Random Forest models. To our surprise, the Random Forest models we implemented did not perform as well as we had hoped. Our best performing Random Forest model exhibited accuracy, precision and recall scores of .8, .9 and .55, respectively (Appendix 4.A). This particular model did offer some excellent insight in the form of the feature importance plot. From this plot we found that the features “NWG” and “MWG” had the heaviest influence on the models prediction(s). The features “NDIMC” and “MDIMC” also had some notable influence on the predictions but much less than the two aforementioned features. We would go on to further explore the reasoning behind this later in the project.

Lastly, a Logistic Regression model was implemented which offered moderate performance. This particular model exhibited scores of .84, .82 and .75 for accuracy, precision and recall, respectively (Appendix 4.C). At this point, we were satisfied with the variety and number of models we had implemented and chose to move forward onto selecting the top three performing models for deployment and evaluation on the test set.

After reviewing all the performance metrics of the various models we concluded that we would deploy the SVM model utilizing an RBF kernel, Gini Decision Tree model and the second Neural Network on the test set and evaluate their performance.

Once deployed on the test set, the SVM with an RBF kernel exhibited similar performance as before with scores of .94, .96 and .89 for accuracy, precision and recall respectively (Appendix R.A). The Decision Tree with Gini also performed exceptionally well on the test set with accuracy, precision and recall scores of .98, .97 and .97 (Appendix R.B). Similarly, the Neural Network offered scores of .97, .97 and .96 for accuracy, precision and recall when evaluated on the test set (Appendix R.C). Overall, these three particular models all performed very well when deployed on the test set.

Within the context of a distributed computing environment and with the potential for our model to be deployed within an online learning system that is incrementally trained, we felt that training time was an important factor to consider for our final model. As such, the Decision Tree with Gini as the impurity measure was by far the most efficient model to train in terms of computational time, amongst all the models we had examined. Given the exceptional performance of this model as well, it is for these reasons that we chose the Decision Tree model as our overall best model.

Throughout the duration of the project we came across a number of intriguing observations. Both the Decision Tree models and Neural Network models exhibited very similar performance. We hypothesized that this was due to the fact that both models aim to emulate human-decision making to a certain extent, albeit with very different approaches. Combine this factor with the dataset being somewhat simple in terms of complexity and it is clear why these models were able to achieve such excellent results. Conversely, to our surprise, the Random

Forest model performed particularly poorly. As this is typically a “go-to” model for most machine learning applications that generally yields decent performance, we hypothesized that the dataset’s low level of complexity and the model splitting on random subsets of features may be doing more harm than good in this case. We were also surprised to see that the Pasting and Bagging models exhibited the exact same performance despite one being performed with replacement and one being performed without replacement.

Most importantly, having discovered that the “NWG” and “MWG” features had a heavy influence on the predictions of the Random Forest prompted us to conduct some further research. These particular features represent the per-matrix 2D tiling value for the GPU kernel. Available values for this particular GPU kernel were sixteen, thirty two, sixty four and one hundred twenty eight. The technique of tiling serves to “reduce global memory accesses by taking advantage of the shared memory on the GPU” and the parallel nature of matrix multiplication (Xu, Penny). From our research we found that with tiling enabled for a GPU the number of “global memory accesses is half the global memory accesses without tiling” enabled (Xu, Penny). As such, with a tiling value of N x N, “potential memory reduction of global memory traffic would be N” (Xu, Penny). Taking this into consideration it is easy to see why tiling and the tiling values assigned to the kernel have such a profound effect on the speed of the matrix multiplication computations within the dataset. To further our understanding of tiling and how it really works under the hood we were able to find a very helpful animated diagram that goes through the process as well (Appendix V.A).

In conclusion, this project offered all group members the opportunity to further explore a variety of machine learning methods and hone their skills in areas such as exploratory data analysis, preprocessing data, as well as data visualization(s). With the use of GPUs becoming ever more prominent within many areas of the tech industry we felt this dataset could offer some very interesting observations and it did not disappoint. We found the impact of tiling and its incredible importance to be of particular interest and feel that we gained a great deal of knowledge from further researching this technique. Overall, we are all satisfied with the results of the project and the experience we gained throughout the course of the project and its many aspects.

References:

- Dataset:

https://archive.ics.uci.edu/ml/datasets/SGEMM+GPU+kernel+performance

C. Nugteren and V. Codreanu, "CLTune: A Generic Auto-Tuner for OpenCL Kernels," *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, Turin, Italy, 2015, pp. 195-202, doi: 10.1109/MCSoC.2015.10.

- Tiled Matrix Multiplication Research:

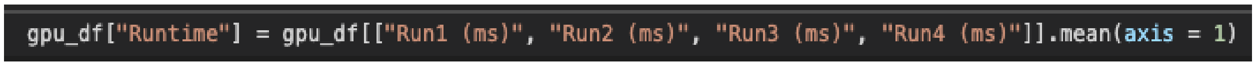
Xu, Penny. *Tiled Matrix Multiplication*. penny-xu.github.io/blog/tiled-matrix-multiplication.

Appendix:

- Items prefixed with 1 were completed by Kyle Calabro
- Items prefixed with 2 were completed by Miguel Esteban Diaz
- Items prefixed with 3 were completed by Mikhail Delyusto
- Items prefixed with 4 were completed by Joana Dervishaj
- Items prefixed with R refer to results of deploying models on the test set
- Items prefixed with V are additional visualizations

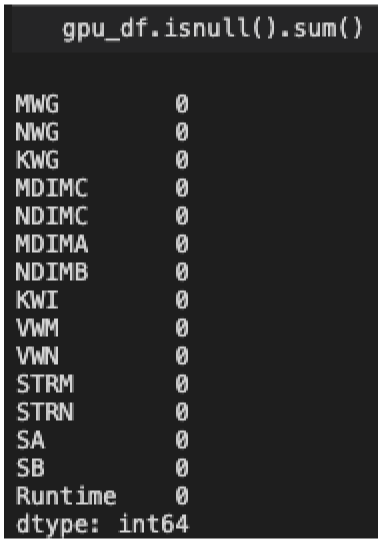
Exploratory Data Analysis and Preprocessing:

1.A) Averaging the Four Given Runtimes



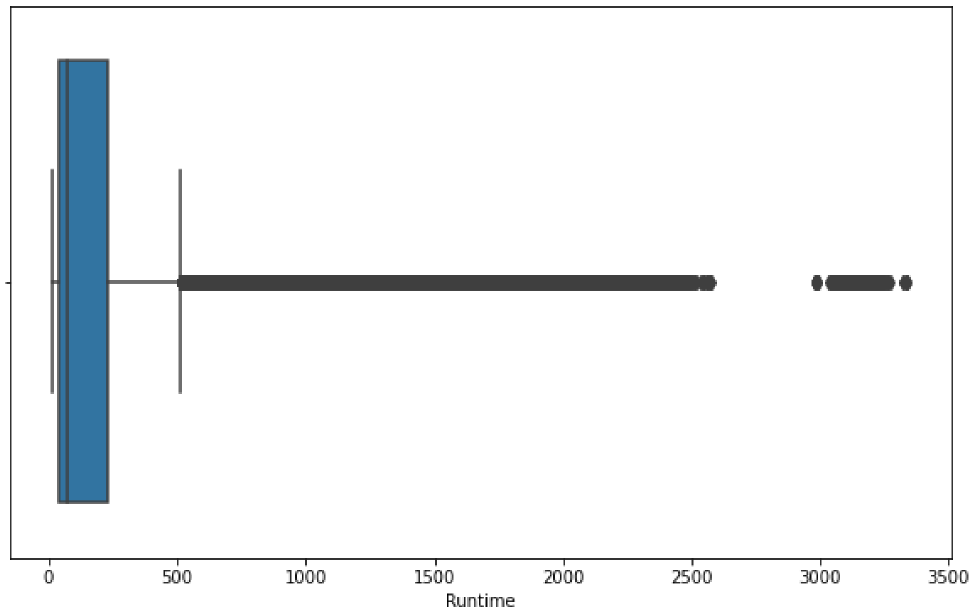
```
gpu_df["Runtime"] = gpu_df[["Run1 (ms)", "Run2 (ms)", "Run3 (ms)", "Run4 (ms)"]].mean(axis = 1)
```

1.B) Checking for Missing Values

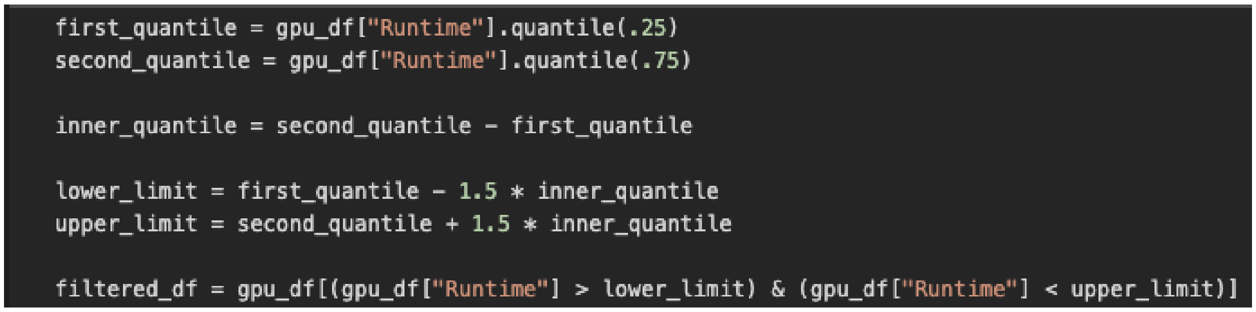


```
   gpu_df.isnull().sum()

MWG        0
NWG        0
KWG        0
MDIMC      0
NDIMC      0
MDIMA      0
NDIMB      0
KWI        0
VWM        0
VWN        0
STRM       0
STRN       0
SA         0
SB         0
Runtime    0
dtype: int64
```

1.C) Outlier Recognition



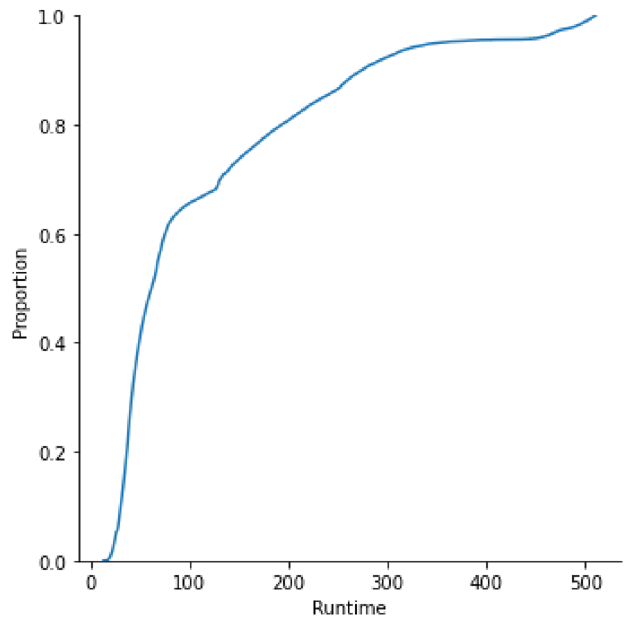
1.D) Resolving the Outliers



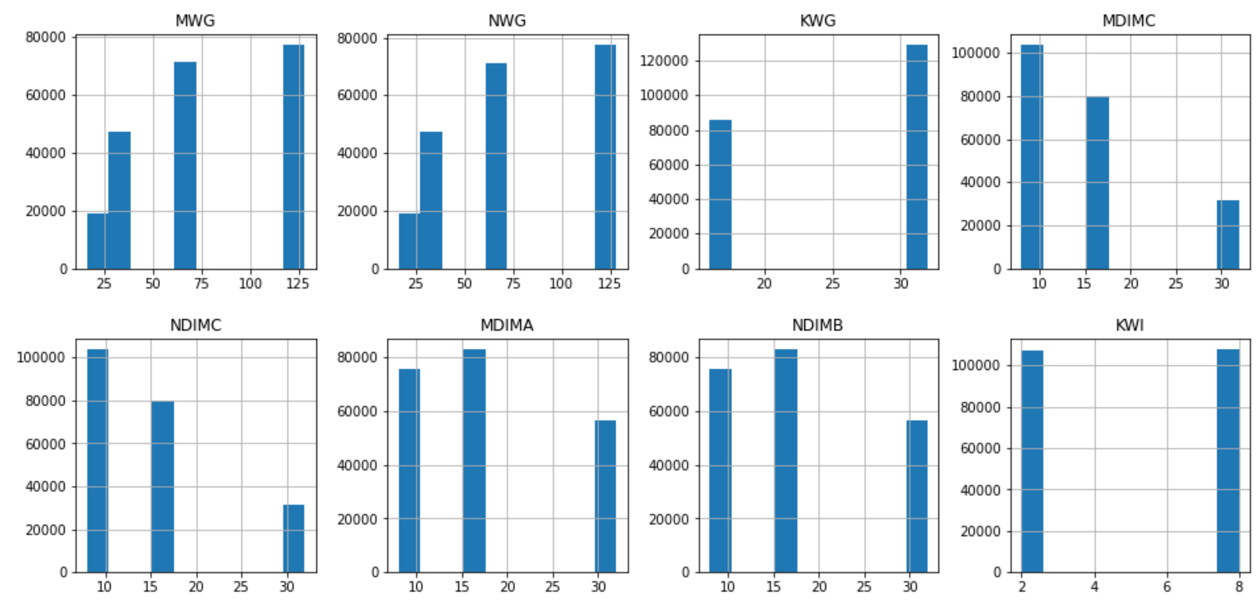
```
first_quantile = gpu_df["Runtime"].quantile(.25)
second_quantile = gpu_df["Runtime"].quantile(.75)

inner_quantile = second_quantile - first_quantile

lower_limit = first_quantile - 1.5 * inner_quantile
upper_limit = second_quantile + 1.5 * inner_quantile

filtered_df = gpu_df[(gpu_df["Runtime"] > lower_limit) & (gpu_df["Runtime"] < upper_limit)]
```

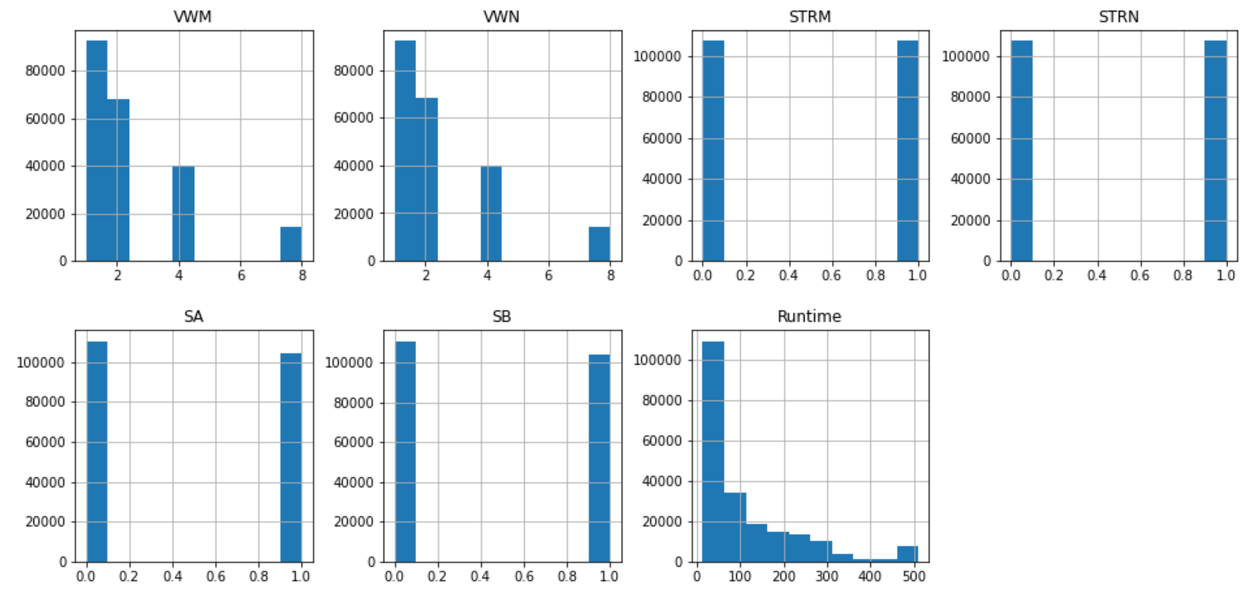
1.E) Empirical Cumulative Distribution Plot Displaying Result of Resolving Outliers

- Height of the curve reflects the proportion of observations with a smaller value

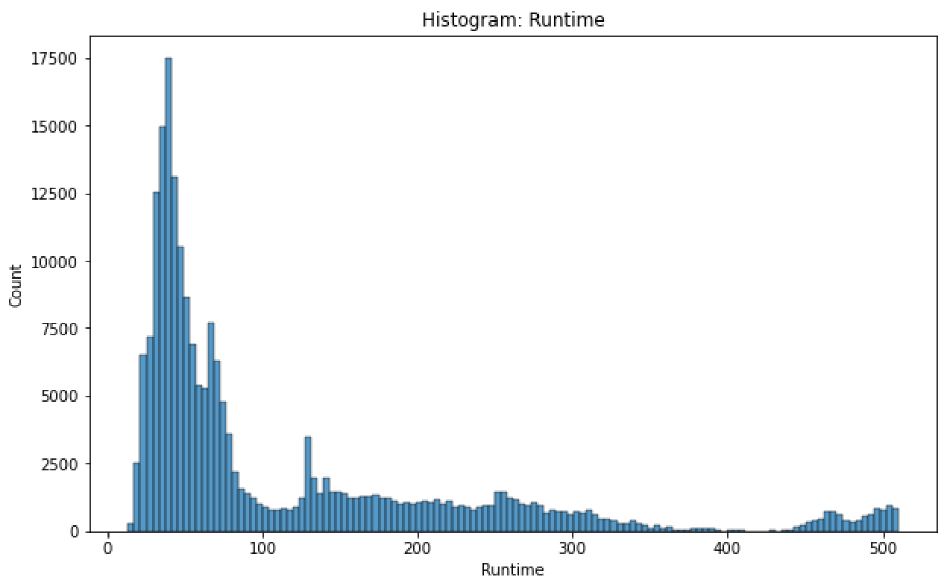


1.F) Histograms of all Features





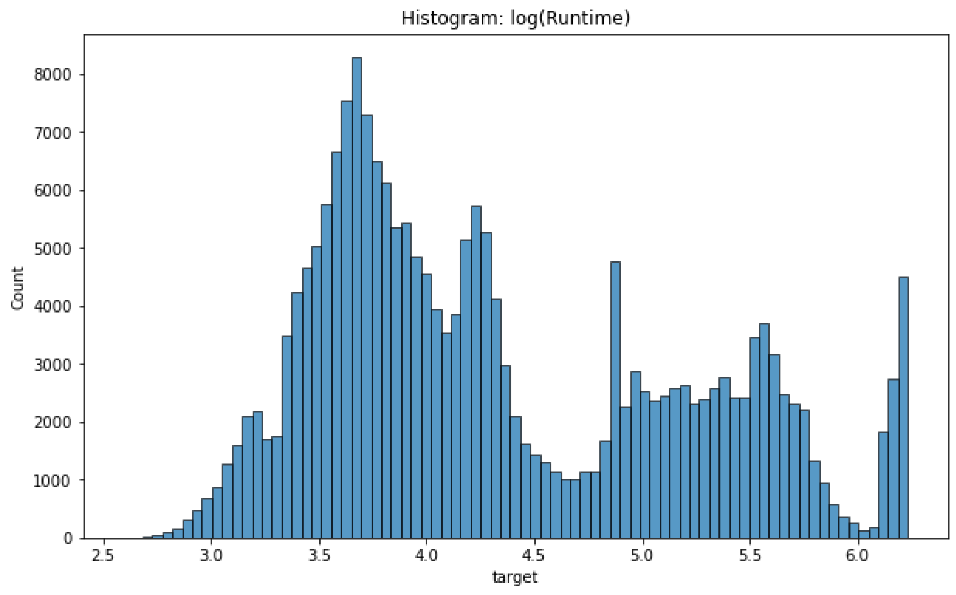
1.G) Histogram of the Runtime Feature



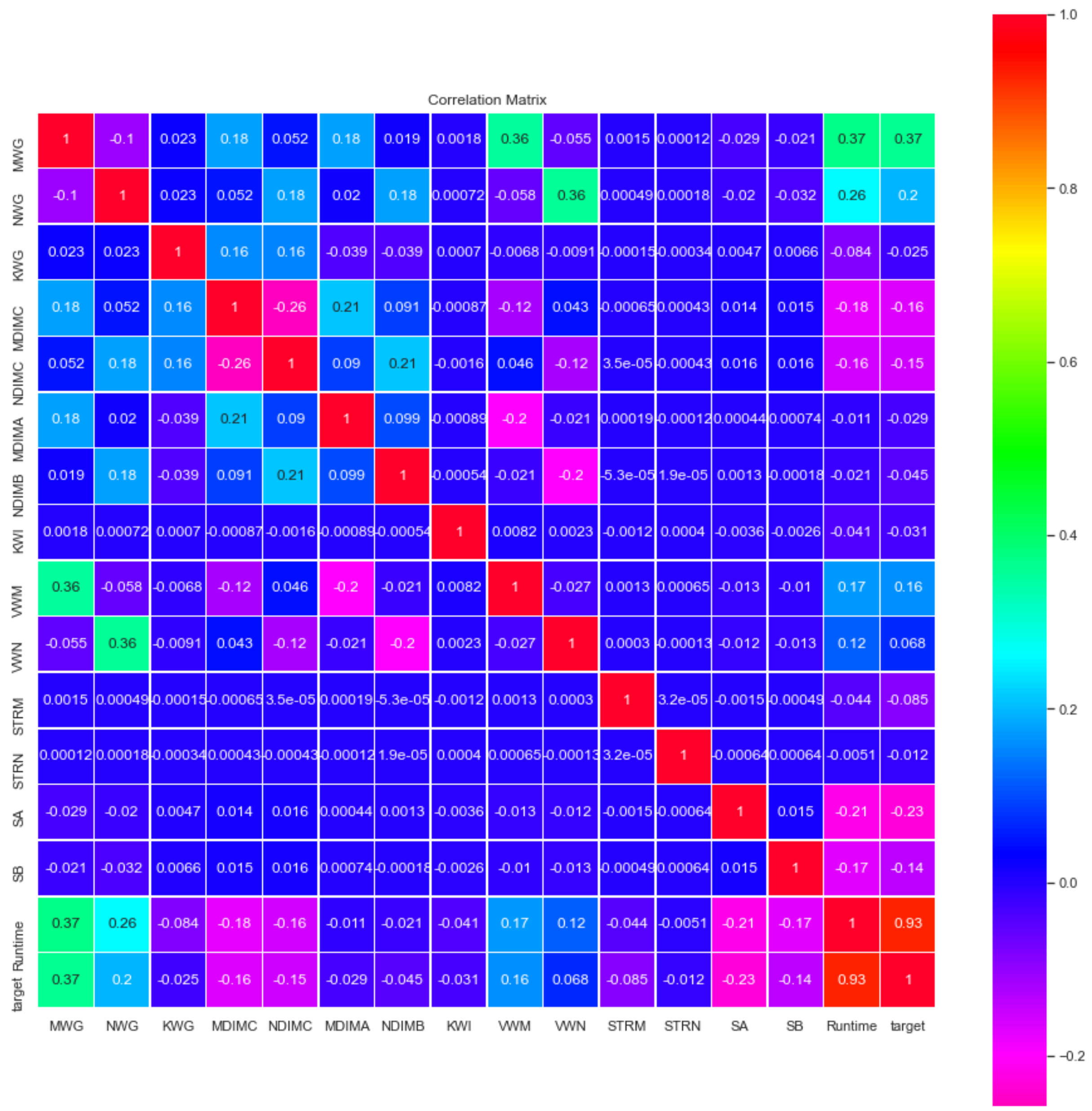
1.H) Transforming Runtime Feature via the Natural-Log

```
filtered_df["target"] = np.log(filtered_df["Runtime"])
```

1.I) Histogram of the Transformed Runtime Feature



1.J) Correlation Matrix

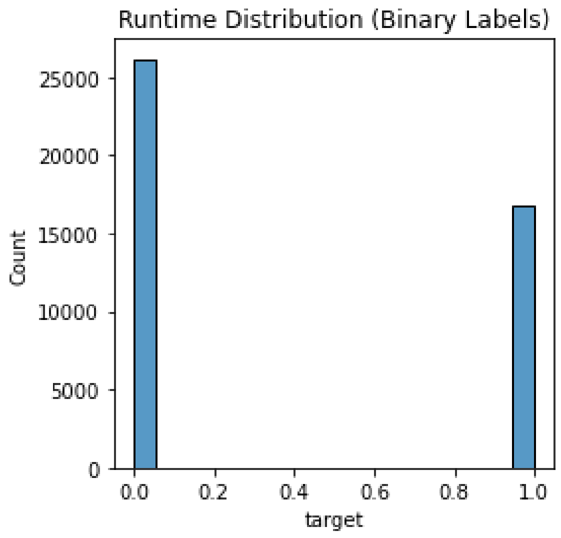


	MWG	NWG	KWG	MDIMC	NDIMC	MDIMA	NDIMB	KWI	VWM	VWN	STRM	STRN	SA	SB	Runtime	target
MWG	1	-0.1	0.023	0.18	0.052	0.18	0.019	0.0018	0.36	-0.055	0.0015	0.00012	-0.029	-0.021	0.37	0.37
NWG	-0.1	1	0.023	0.052	0.18	0.02	0.18	0.00072	-0.058	0.36	0.00049	0.00018	-0.02	-0.032	0.26	0.2
KWG	0.023	0.023	1	0.16	0.16	-0.039	-0.039	0.0007	-0.0068	-0.0091	-0.00015	-0.00034	0.0047	0.0066	-0.084	-0.025
MDIMC	0.18	0.052	0.16	1	-0.26	0.21	0.091	-0.00087	-0.12	0.043	-0.00065	0.00043	0.014	0.015	-0.18	-0.16
NDIMC	0.052	0.18	0.16	-0.26	1	0.09	0.21	-0.0016	0.046	-0.12	3.5e-05	-0.00043	0.016	0.016	-0.16	-0.15
MDIMA	0.18	0.02	-0.039	0.21	0.09	1	0.099	-0.00089	-0.2	-0.021	0.00019	-0.00012	0.00044	0.00074	-0.011	-0.029
NDIMB	0.019	0.18	-0.039	0.091	0.21	0.099	1	-0.00054	-0.021	-0.2	-5.3e-05	1.9e-05	0.0013	-0.00018	-0.021	-0.045
KWI	0.0018	0.00072	0.0007	-0.00087	-0.0016	-0.00089	-0.00054	1	0.0082	0.0023	-0.0012	0.0004	-0.0036	-0.0026	-0.041	-0.031
VWM	0.36	-0.058	-0.0068	-0.12	0.046	-0.2	-0.021	0.0082	1	-0.027	0.0013	0.00065	-0.013	-0.01	0.17	0.16
VWN	-0.055	0.36	-0.0091	0.043	-0.12	-0.021	-0.2	0.0023	-0.027	1	0.0003	-0.00013	-0.012	-0.013	0.12	0.068
STRM	0.0015	0.00049	-0.00015	-0.00065	3.5e-05	0.00019	-5.3e-05	-0.0012	0.0013	0.0003	1	3.2e-05	-0.0015	-0.00049	-0.044	-0.085
STRN	0.00012	0.00018	-0.00034	0.00043	-0.00043	-0.00012	1.9e-05	0.0004	0.00065	-0.00013	3.2e-05	1	-0.00064	0.00064	-0.0051	-0.012
SA	-0.029	-0.02	0.0047	0.014	0.016	0.00044	0.0013	-0.0036	-0.013	-0.012	-0.0015	-0.00064	1	0.015	-0.21	-0.23
SB	-0.021	-0.032	0.0066	0.015	0.016	0.00074	-0.00018	-0.0026	-0.01	-0.013	-0.00049	0.00064	0.015	1	-0.17	-0.14
Runtime	0.37	0.26	-0.084	-0.18	-0.16	-0.011	-0.021	-0.041	0.17	0.12	-0.044	-0.0051	-0.21	-0.17	1	0.93
target	0.37	0.2	-0.025	-0.16	-0.15	-0.029	-0.045	-0.031	0.16	0.068	-0.085	-0.012	-0.23	-0.14	0.93	1

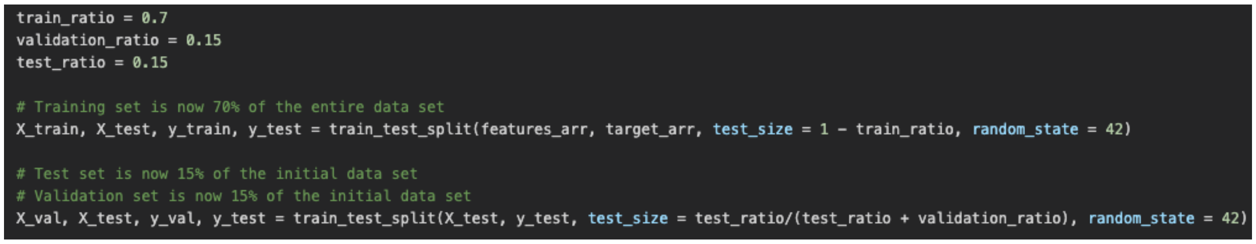
1.K) Converting Runtime to a Binary Value

```
mean = filtered_df["target"].mean()
filtered_df.loc[filtered_df["target"] <= mean, "target"] = 0 # Fast runtime
filtered_df.loc[filtered_df["target"] > mean, "target"] = 1 # Slow runtime

target_arr = filtered_df["target"].values
target_arr = target_arr.astype(int)

features_arr = filtered_df.drop(columns = ["target", "Runtime"], axis = 1).values
```

1.L) Distribution of Runtime as a Binary Value



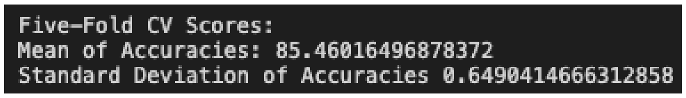
1.M) Training, Validation and Test Set Splitting



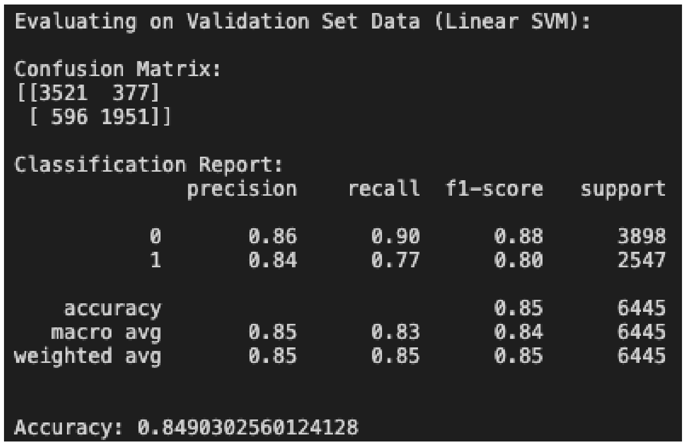
```
train_ratio = 0.7
validation_ratio = 0.15
test_ratio = 0.15

# Training set is now 70% of the entire data set
X_train, X_test, y_train, y_test = train_test_split(features_arr, target_arr, test_size = 1 - train_ratio, random_state = 42)

# Test set is now 15% of the initial data set
# Validation set is now 15% of the initial data set
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size = test_ratio/(test_ratio + validation_ratio), random_state = 42)
```

<u>Exploring Models with the Validation Set:</u>

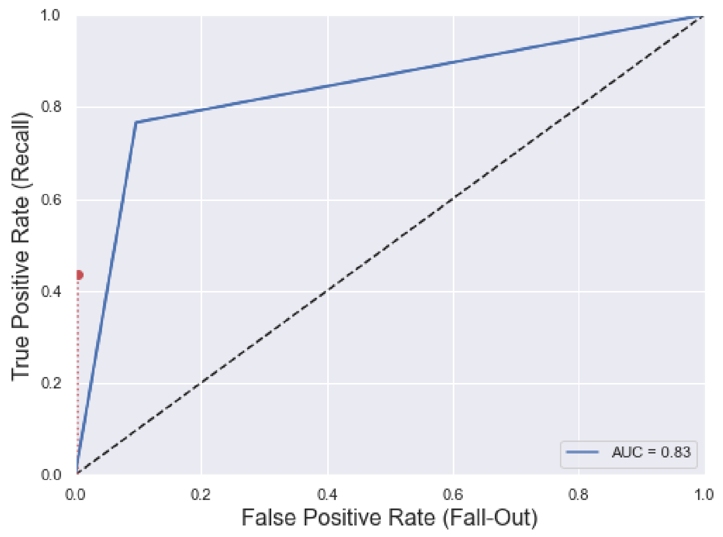
1.N) Support Vector Machine with Linear Kernel



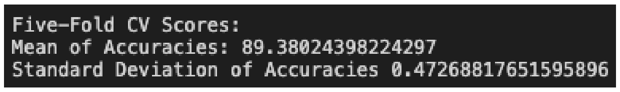
```
Five-Fold CV Scores:
Mean of Accuracies: 85.46016496878372
Standard Deviation of Accuracies 0.6490414666312858
```



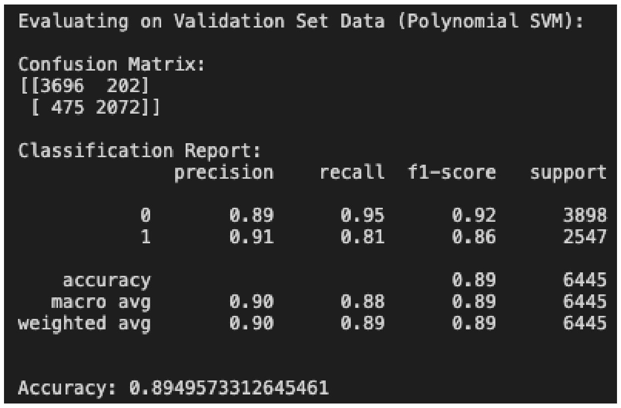
```
Evaluating on Validation Set Data (Linear SVM):

Confusion Matrix:
[[3521  377]
 [ 596 1951]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      3898
           1       0.84      0.77      0.80      2547

    accuracy                           0.85      6445
   macro avg       0.85      0.83      0.84      6445
weighted avg       0.85      0.85      0.85      6445


Accuracy: 0.8490302560124128
```



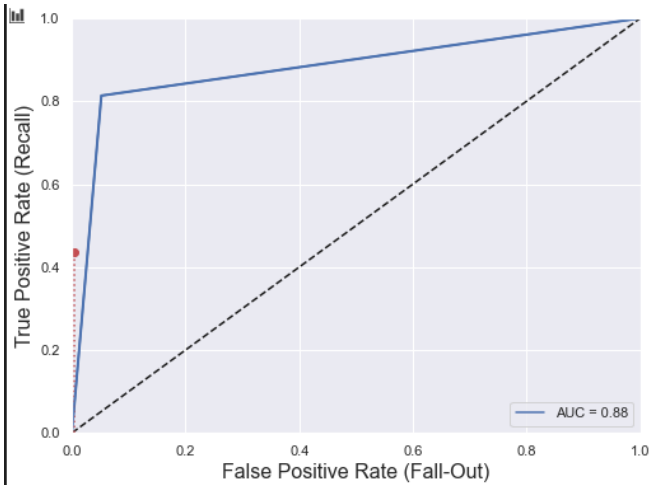
1.O) Support Vector Machine with Polynomial Kernel



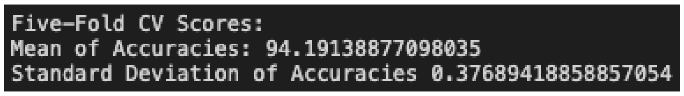
```
Five-Fold CV Scores:
Mean of Accuracies: 89.38024398224297
Standard Deviation of Accuracies 0.47268817651595896
```



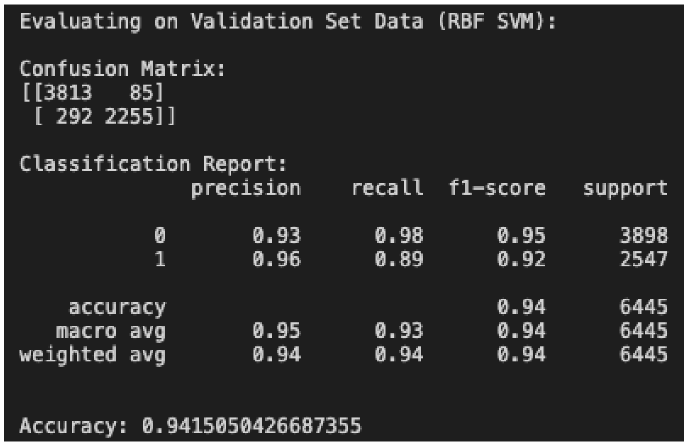
```
Evaluating on Validation Set Data (Polynomial SVM):

Confusion Matrix:
[[3696  202]
 [ 475 2072]]

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      3898
           1       0.91      0.81      0.86      2547

    accuracy                           0.89      6445
   macro avg       0.90      0.88      0.89      6445
weighted avg       0.90      0.89      0.89      6445


Accuracy: 0.8949573312645461
```



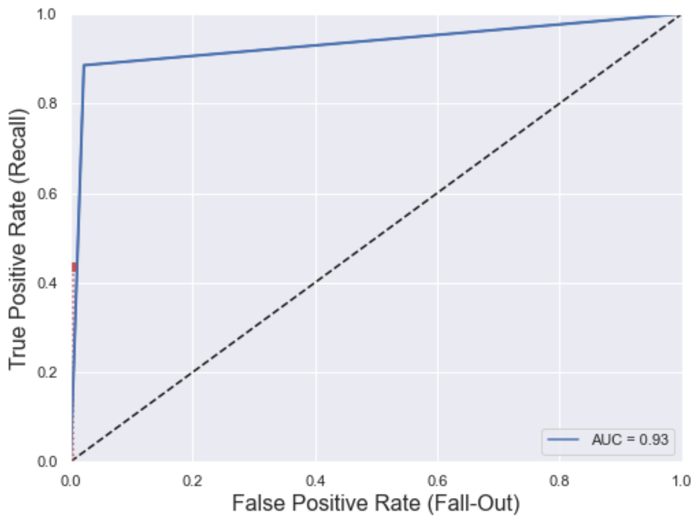
1.P) Support Vector Machine with RBF Kernel



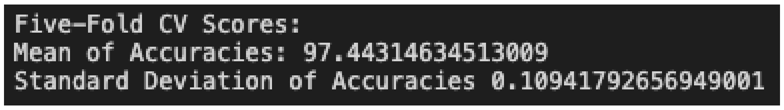
```
Five-Fold CV Scores:
Mean of Accuracies: 94.19138877098035
Standard Deviation of Accuracies 0.37689418858857054
```



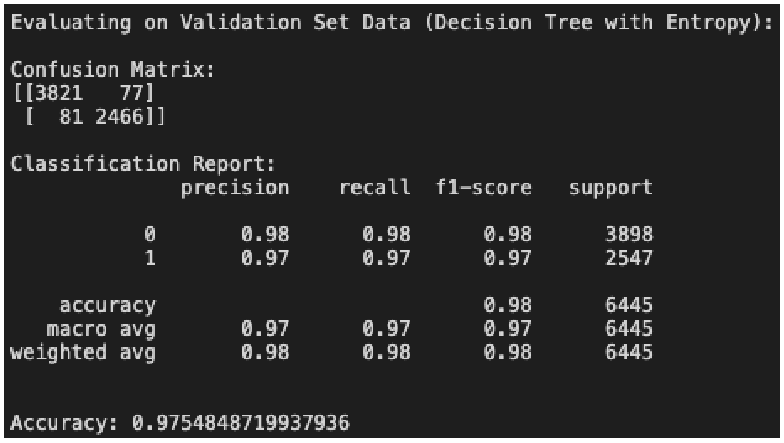
```
Evaluating on Validation Set Data (RBF SVM):

Confusion Matrix:
[[3813   85]
 [ 292 2255]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      3898
           1       0.96      0.89      0.92      2547

    accuracy                           0.94      6445
   macro avg       0.95      0.93      0.94      6445
weighted avg       0.94      0.94      0.94      6445


Accuracy: 0.9415050426687355
```



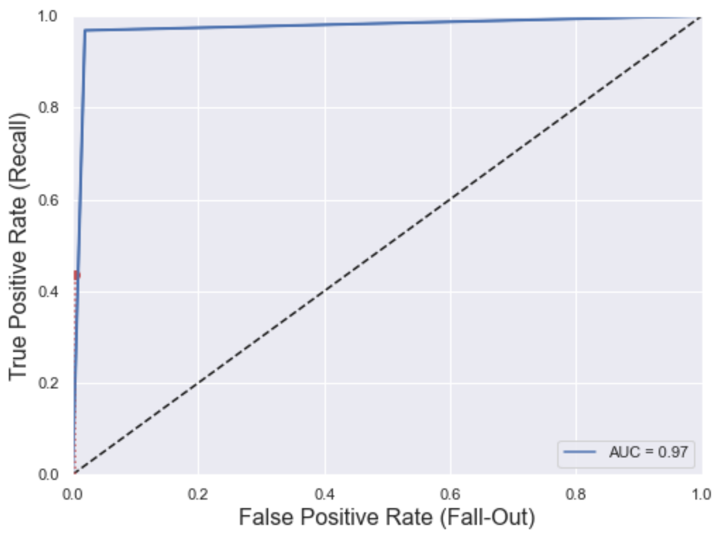
1.Q) Decision Tree with Entropy as Impurity Measure



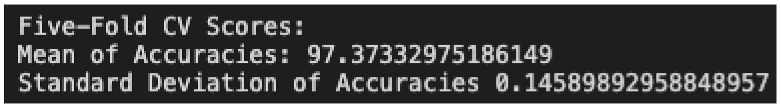
```
Five-Fold CV Scores:
Mean of Accuracies: 97.44314634513009
Standard Deviation of Accuracies 0.10941792656949001
```



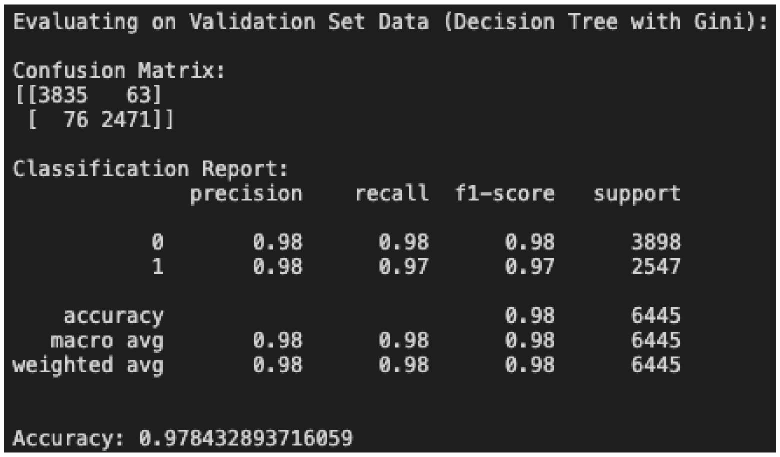
```
Evaluating on Validation Set Data (Decision Tree with Entropy):

Confusion Matrix:
[[3821   77]
 [  81 2466]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      3898
           1       0.97      0.97      0.97      2547

    accuracy                           0.98      6445
   macro avg       0.97      0.97      0.97      6445
weighted avg       0.98      0.98      0.98      6445


Accuracy: 0.9754848719937936
```



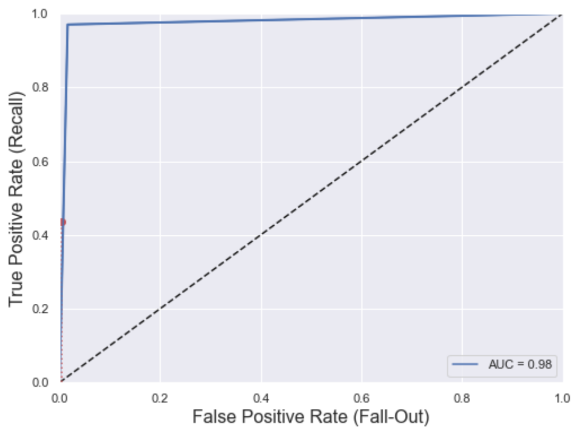
1.R) Decision Tree with Gini as Impurity Measure



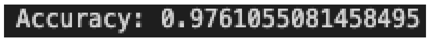
```
Five-Fold CV Scores:
Mean of Accuracies: 97.37332975186149
Standard Deviation of Accuracies 0.14589892958848957
```



```
Evaluating on Validation Set Data (Decision Tree with Gini):

Confusion Matrix:
[[3835   63]
 [  76 2471]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      3898
           1       0.98      0.97      0.97      2547

    accuracy                           0.98      6445
   macro avg       0.98      0.98      0.98      6445
weighted avg       0.98      0.98      0.98      6445


Accuracy: 0.978432893716059
```

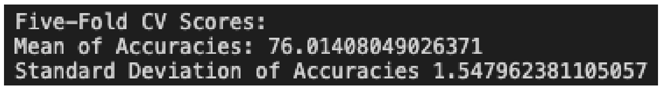


1.S) Pruned Decision Tree with Gini as Impurity Measure

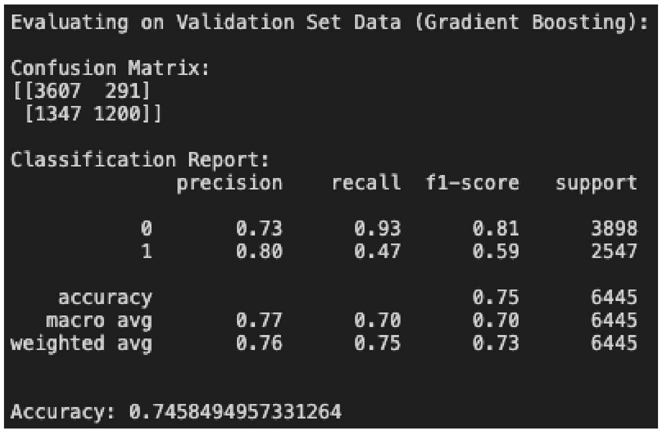


```
Accuracy: 0.9761055081458495
```

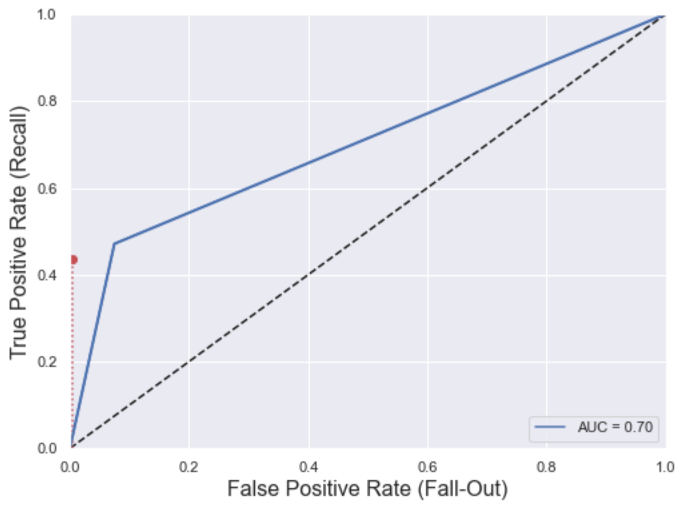
1.T) Gradient Boosting



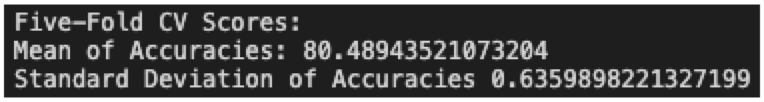
```
Five-Fold CV Scores:
Mean of Accuracies: 76.01408049026371
Standard Deviation of Accuracies 1.547962381105057
```



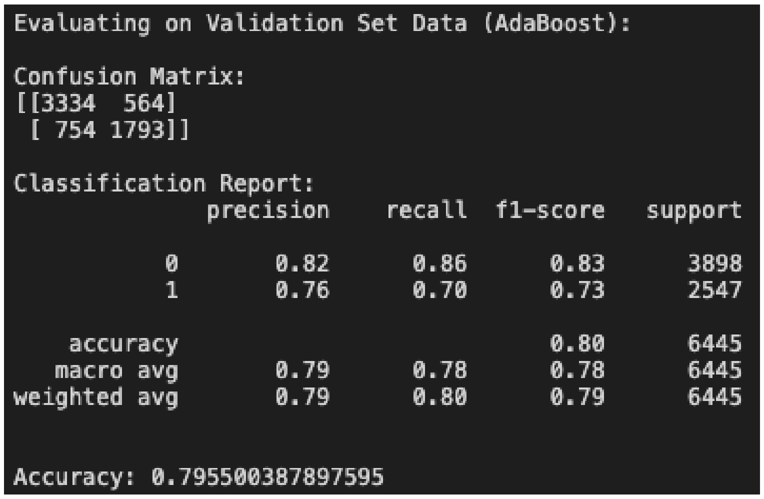
```
Evaluating on Validation Set Data (Gradient Boosting):

Confusion Matrix:
[[3607  291]
 [1347 1200]]

Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.93      0.81      3898
           1       0.80      0.47      0.59      2547

    accuracy                           0.75      6445
   macro avg       0.77      0.70      0.70      6445
weighted avg       0.76      0.75      0.73      6445


Accuracy: 0.7458494957331264
```



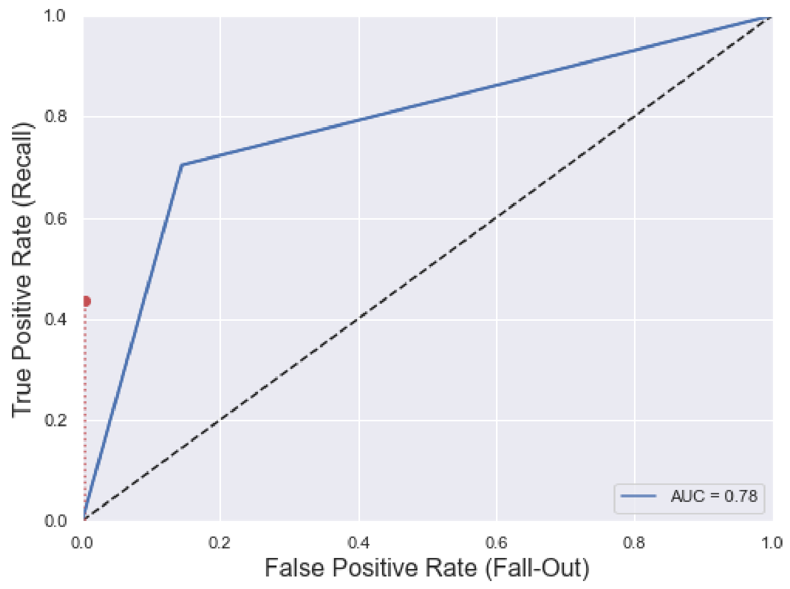
1.U) AdaBoost



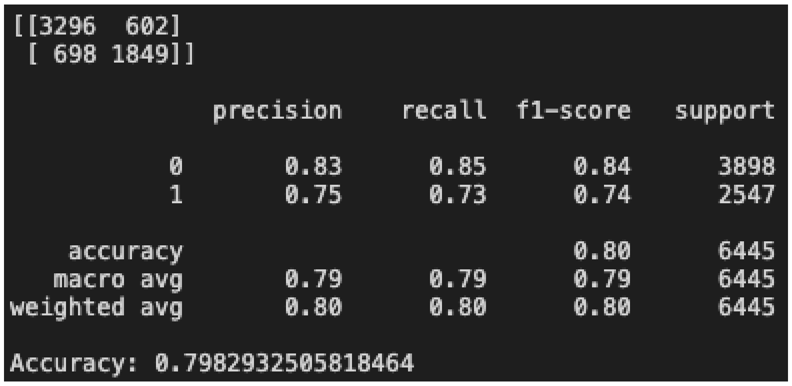
```
Five-Fold CV Scores:
Mean of Accuracies: 80.48943521073204
Standard Deviation of Accuracies 0.6359898221327199
```



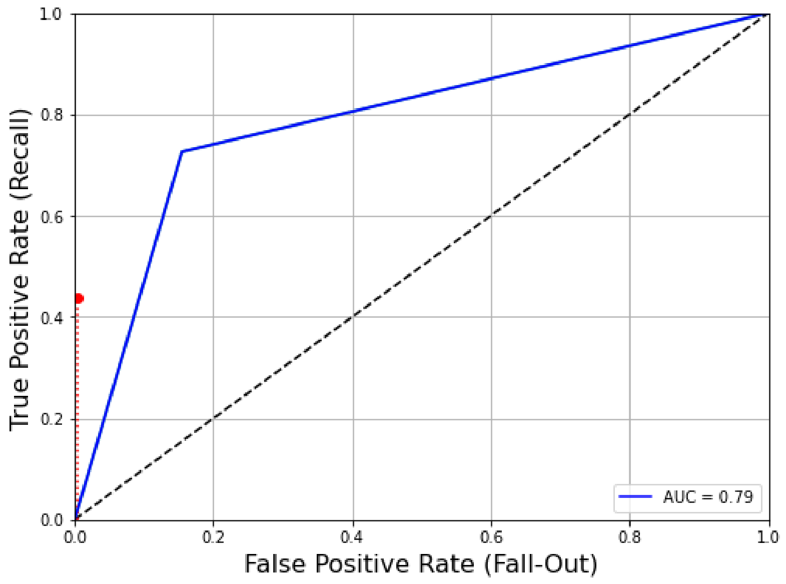
```
Evaluating on Validation Set Data (AdaBoost):

Confusion Matrix:
[[3334  564]
 [ 754 1793]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.86      0.83      3898
           1       0.76      0.70      0.73      2547

    accuracy                           0.80      6445
   macro avg       0.79      0.78      0.78      6445
weighted avg       0.79      0.80      0.79      6445


Accuracy: 0.795500387897595
```



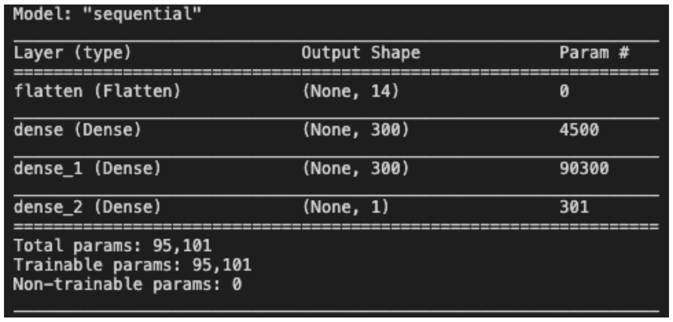
2.A) Stochastic Gradient Descent (SGD)



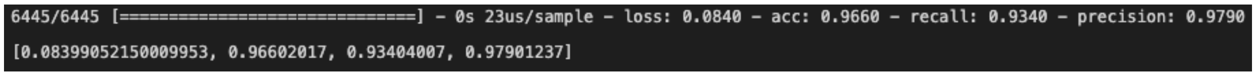
```
[[3296  602]
 [ 698 1849]]

              precision    recall  f1-score   support

           0       0.83      0.85      0.84      3898
           1       0.75      0.73      0.74      2547

    accuracy                           0.80      6445
   macro avg       0.79      0.79      0.79      6445
weighted avg       0.80      0.80      0.80      6445

Accuracy: 0.7982932505818464
```



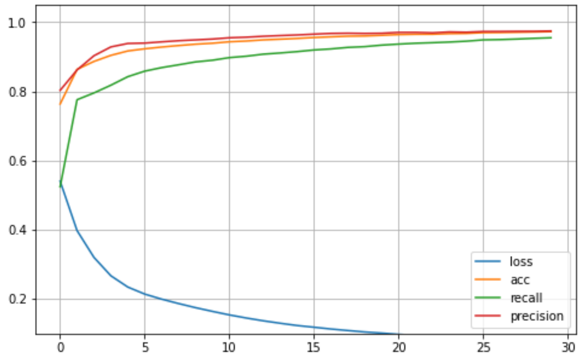
2.B) Mini-Batch Gradient Descent via Neural Network



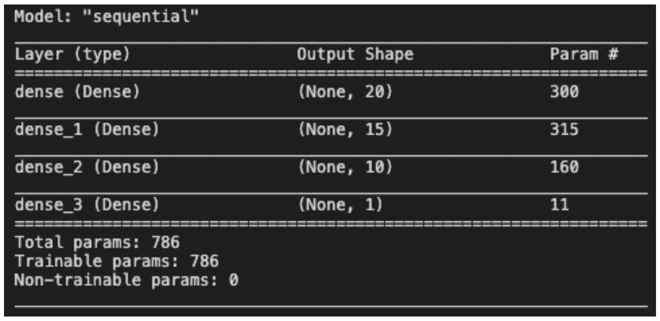
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 14)                0
_________________________________________________________________
dense (Dense)                (None, 300)               4500
_________________________________________________________________
dense_1 (Dense)              (None, 300)               90300
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 301
=================================================================
Total params: 95,101
Trainable params: 95,101
Non-trainable params: 0
_________________________________________________________________
```



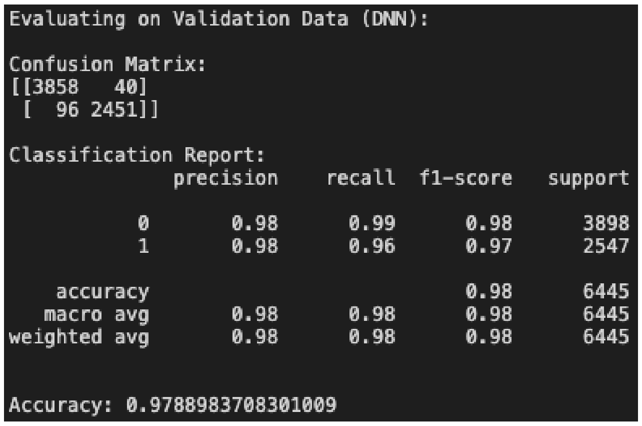
```
6445/6445 [==============================] - 0s 23us/sample - loss: 0.0840 - acc: 0.9660 - recall: 0.9340 - precision: 0.9790
[0.08399052150009953, 0.96602017, 0.93404007, 0.97901237]
```



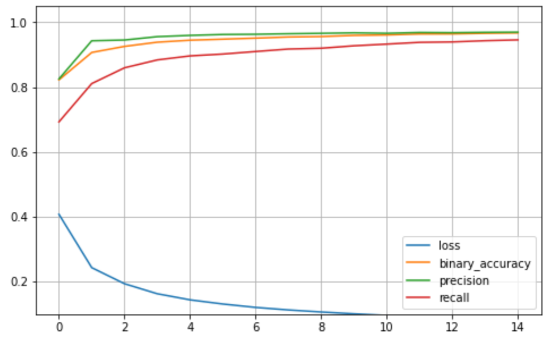
3.A) Neural Network



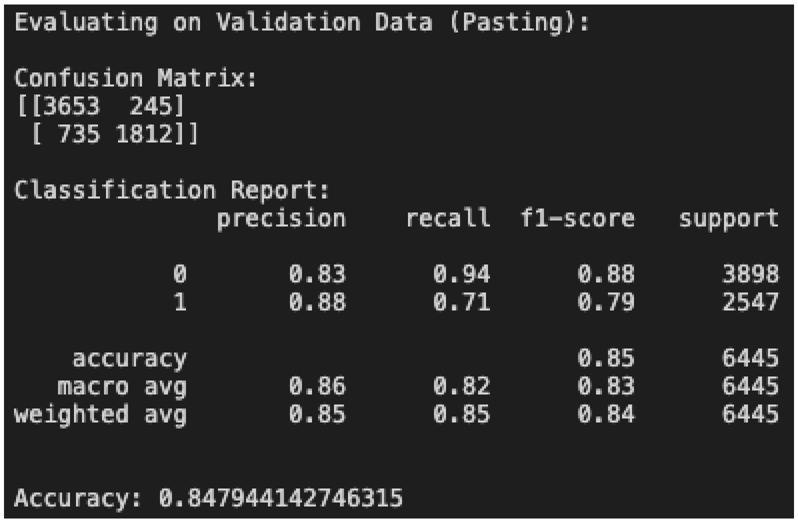
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 20)                300
_________________________________________________________________
dense_1 (Dense)              (None, 15)                315
_________________________________________________________________
dense_2 (Dense)              (None, 10)                160
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 786
Trainable params: 786
Non-trainable params: 0
_________________________________________________________________
```



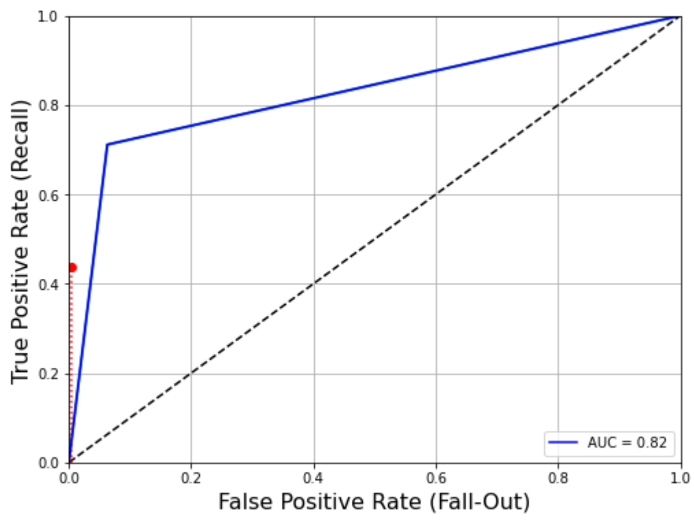
```
Evaluating on Validation Data (DNN):

Confusion Matrix:
[[3858   40]
 [  96 2451]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      3898
           1       0.98      0.96      0.97      2547

    accuracy                           0.98      6445
   macro avg       0.98      0.98      0.98      6445
weighted avg       0.98      0.98      0.98      6445


Accuracy: 0.9788983708301009
```



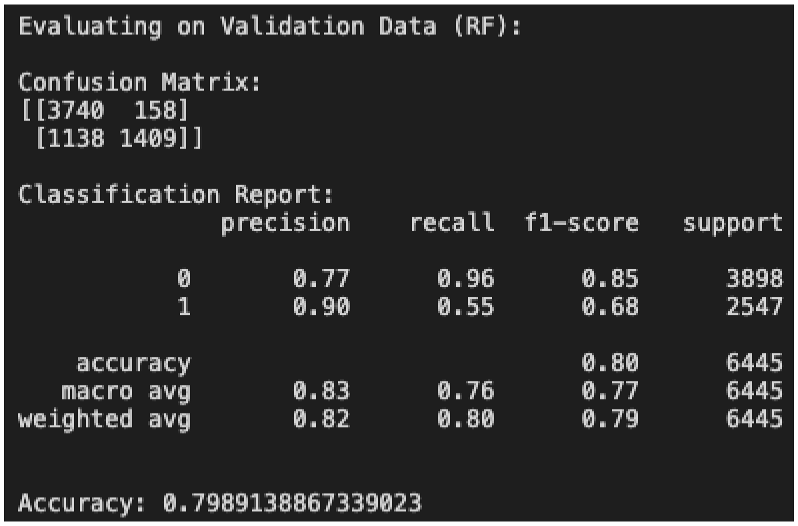
3.B) Pasting



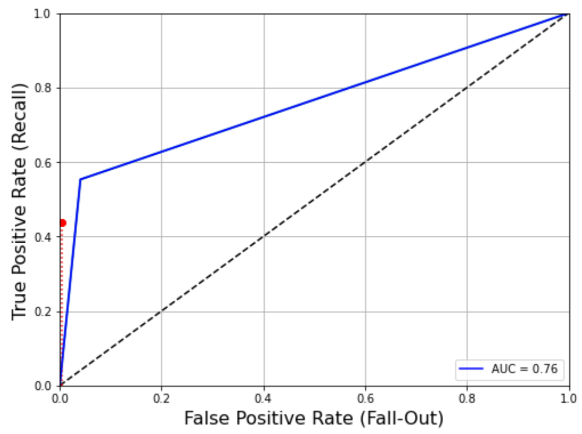
```
Evaluating on Validation Data (Pasting):

Confusion Matrix:
[[3653  245]
 [ 735 1812]]

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.94      0.88      3898
           1       0.88      0.71      0.79      2547

    accuracy                           0.85      6445
   macro avg       0.86      0.82      0.83      6445
weighted avg       0.85      0.85      0.84      6445


Accuracy:  0.847944142746315
```

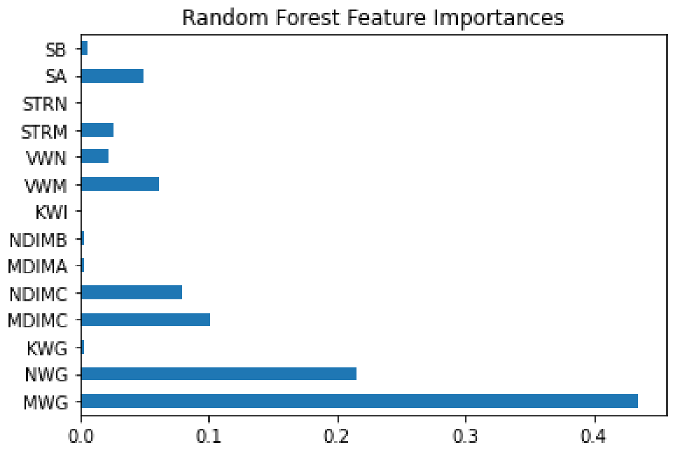


4.A) Random Forest

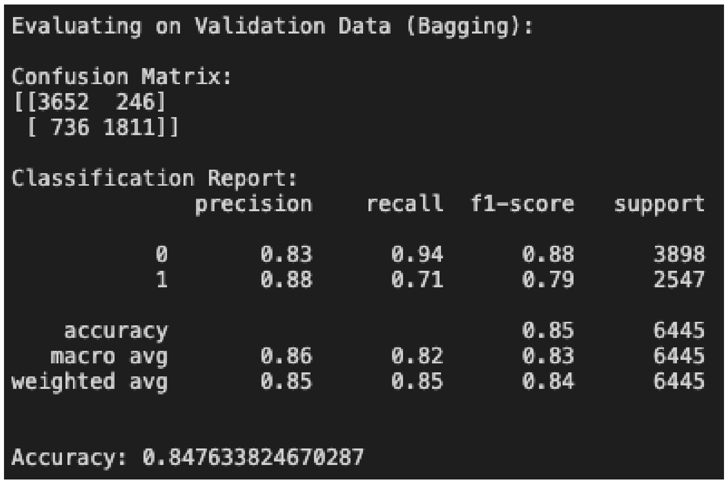


```
Evaluating on Validation Data (RF):

Confusion Matrix:
[[3740  158]
 [1138 1409]]

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.96      0.85      3898
           1       0.90      0.55      0.68      2547

    accuracy                           0.80      6445
   macro avg       0.83      0.76      0.77      6445
weighted avg       0.82      0.80      0.79      6445


Accuracy:  0.7989138867339023
```

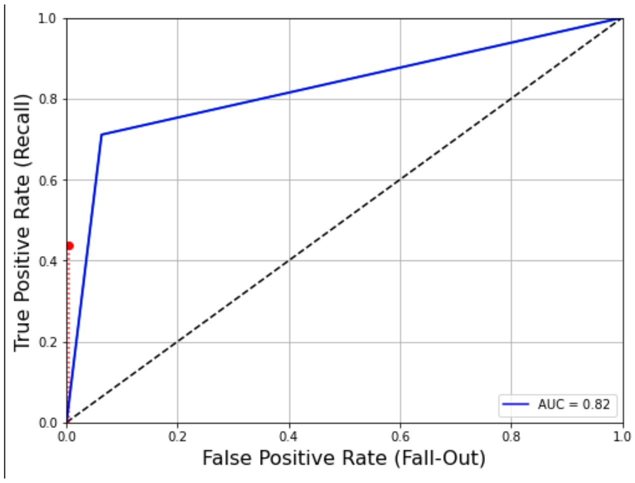




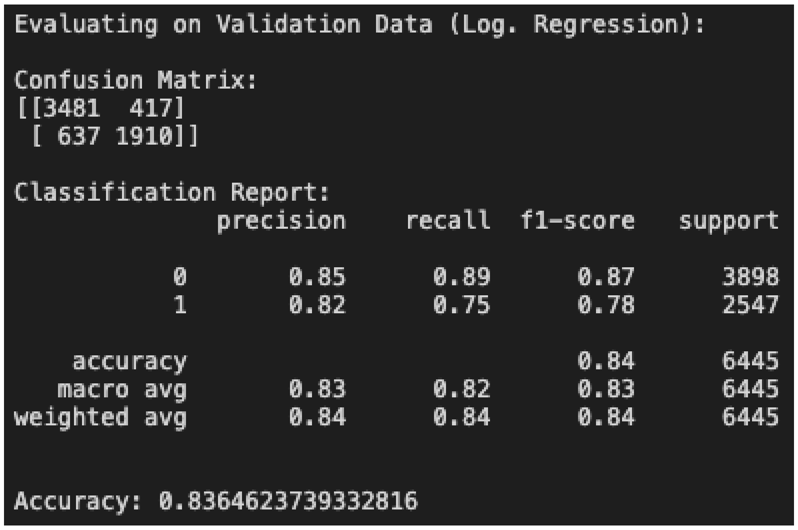
4.B) Bagging



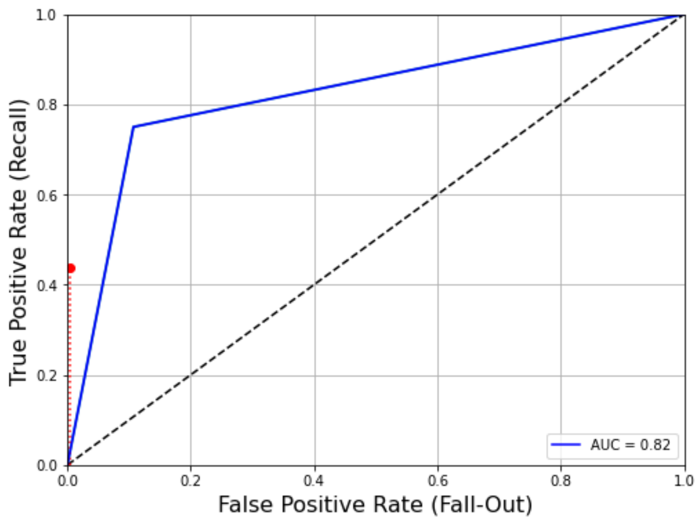
```
Evaluating on Validation Data (Bagging):

Confusion Matrix:
[[3652  246]
 [ 736 1811]]

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.94      0.88      3898
           1       0.88      0.71      0.79      2547

    accuracy                           0.85      6445
   macro avg       0.86      0.82      0.83      6445
weighted avg       0.85      0.85      0.84      6445


Accuracy: 0.847633824670287
```



4.C) Logistic Regression

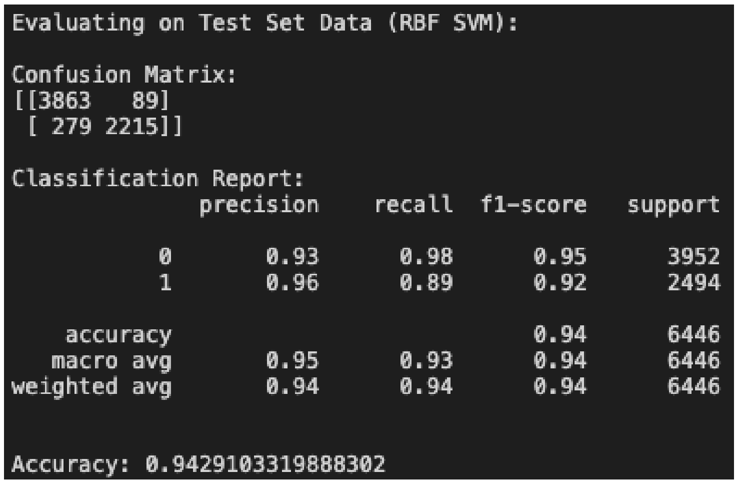


```
Evaluating on Validation Data (Log. Regression):

Confusion Matrix:
[[3481  417]
 [ 637 1910]]

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      3898
           1       0.82      0.75      0.78      2547

    accuracy                           0.84      6445
   macro avg       0.83      0.82      0.83      6445
weighted avg       0.84      0.84      0.84      6445


Accuracy: 0.8364623739332816
```

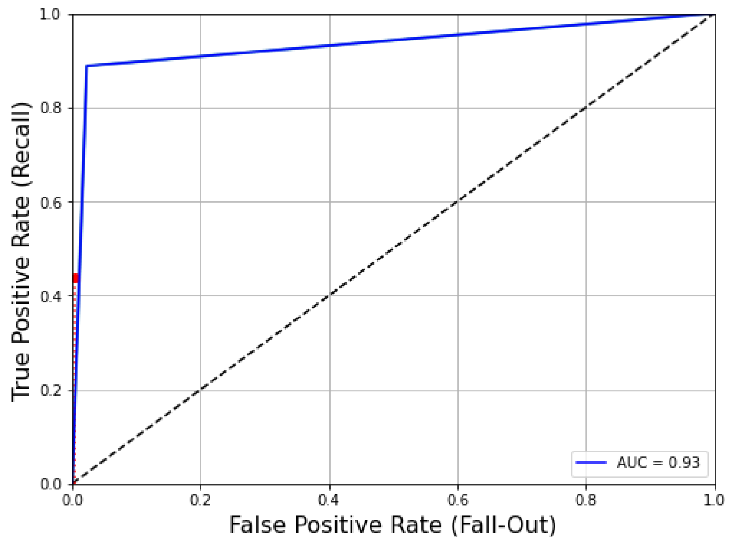


<u>Results of Model Deployment on the Test Set:</u>

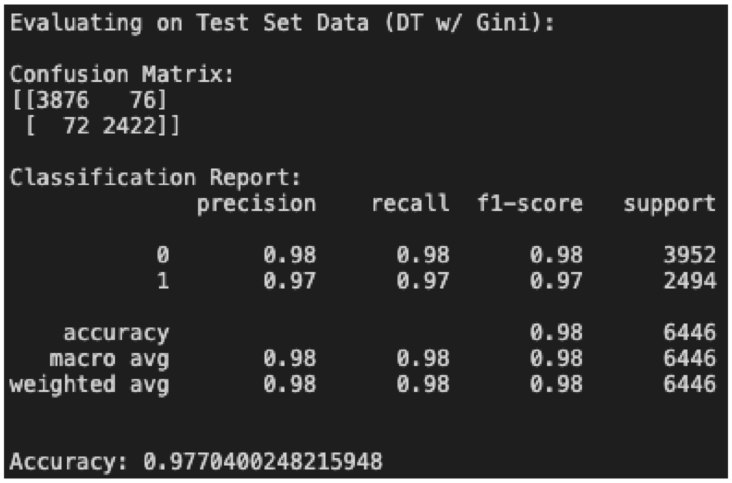
R.A) Support Vector Machine with RBF Kernel



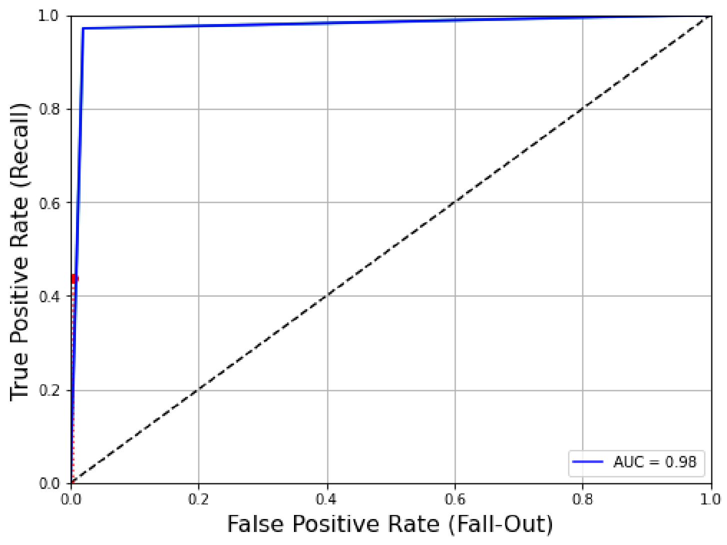
```
Evaluating on Test Set Data (RBF SVM):

Confusion Matrix:
[[3863   89]
 [ 279 2215]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      3952
           1       0.96      0.89      0.92      2494

    accuracy                           0.94      6446
   macro avg       0.95      0.93      0.94      6446
weighted avg       0.94      0.94      0.94      6446


Accuracy: 0.9429103319888302
```



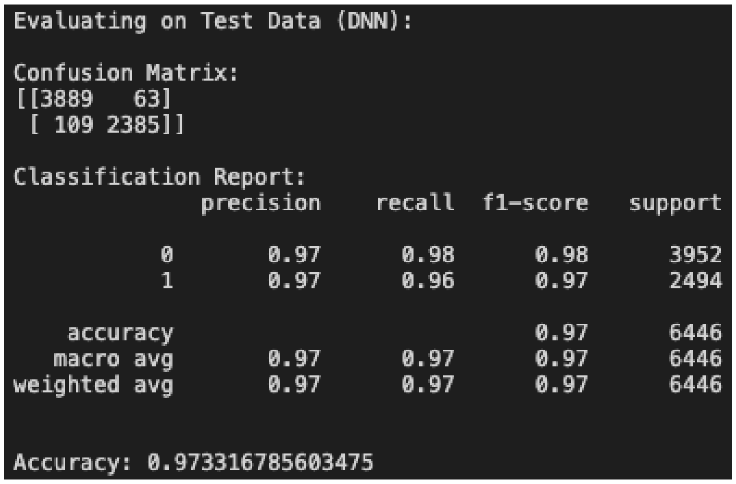
R.B) Decision Tree with Gini



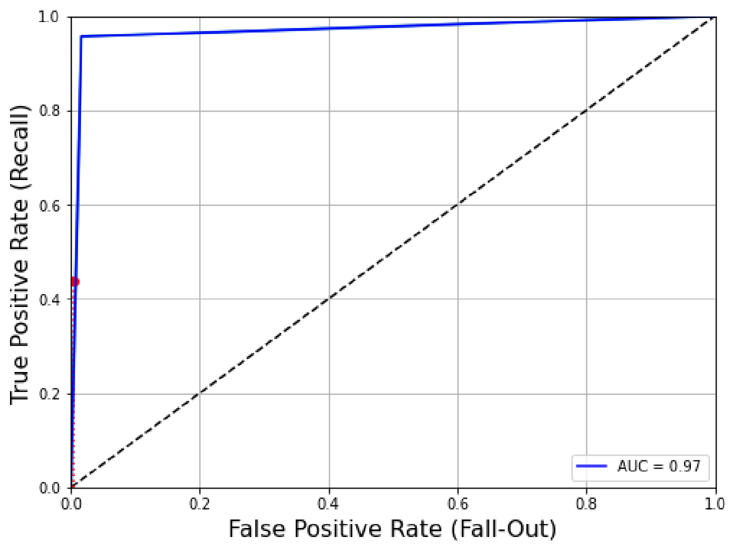
```
Evaluating on Test Set Data (DT w/ Gini):

Confusion Matrix:
[[3876   76]
 [  72 2422]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      3952
           1       0.97      0.97      0.97      2494

    accuracy                           0.98      6446
   macro avg       0.98      0.98      0.98      6446
weighted avg       0.98      0.98      0.98      6446


Accuracy: 0.9770400248215948
```



R.C) Neural Network

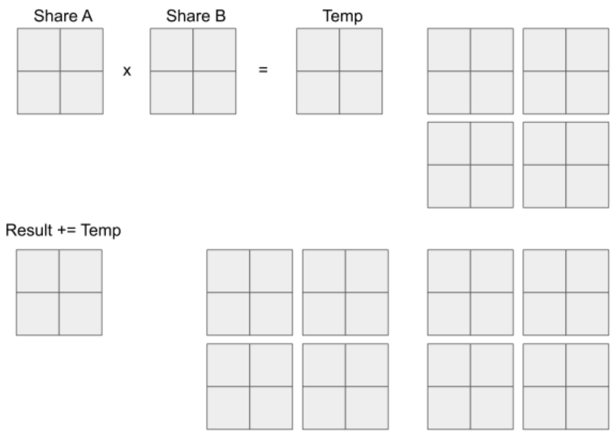


```
Evaluating on Test Data (DNN):

Confusion Matrix:
[[3889   63]
 [ 109 2385]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98      3952
           1       0.97      0.96      0.97      2494

    accuracy                           0.97      6446
   macro avg       0.97      0.97      0.97      6446
weighted avg       0.97      0.97      0.97      6446


Accuracy: 0.973316785603475
```



Other Visualizations:

V.A) Diagram Displaying how Tiling Functions (Xu, Penny)


Share A
x
Share B
=
Temp
Result += Temp