# **Introduction:**

The goal of the project is to predict the run time of various GPU kernel performance time based on metrics on various combinations. Using this data, we would be able to classify the outcome performance of each combination using SVM, Decision Trees, Boosting, cross validation techniques, Artificial Neural Networks and KNN algorithm. By using these classification algorithms, we model the efficient timeframe from various parameters.

# **DataSet:**

The Dataset is downloaded from the UCI machine learning repository. A brief description of the dataset is as follows:

It contains 241600 records on 18 attributes.

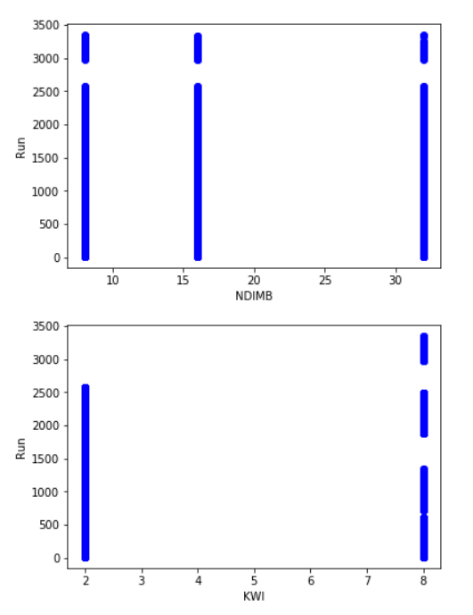
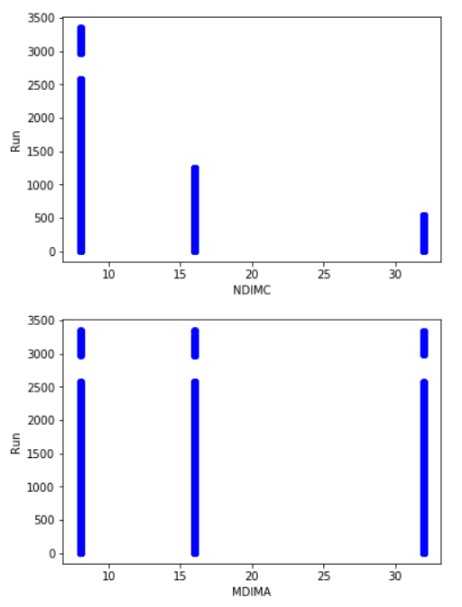
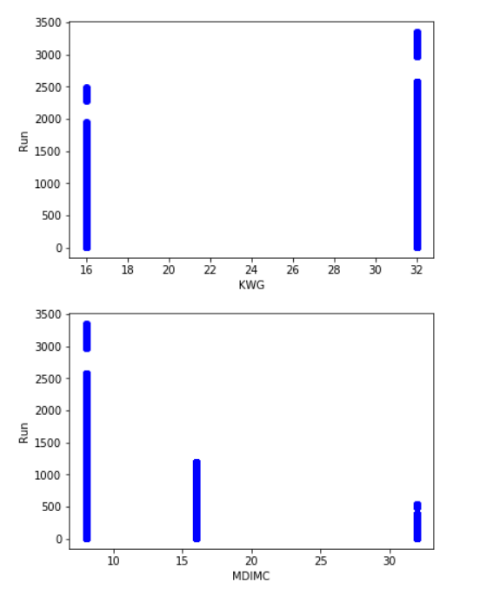
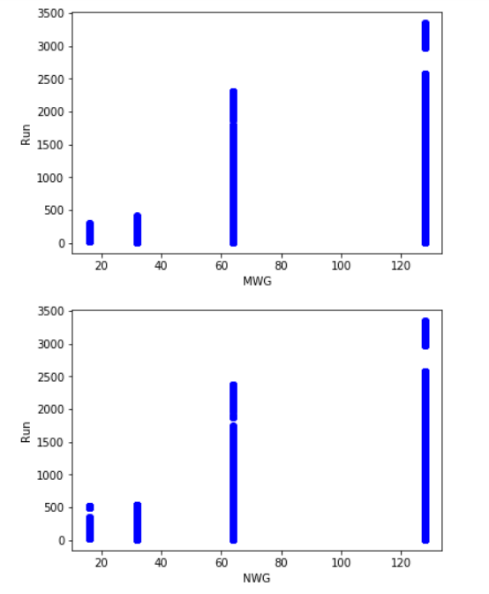
Averaged the runtime value of the 4 columns to a single outcome variable.

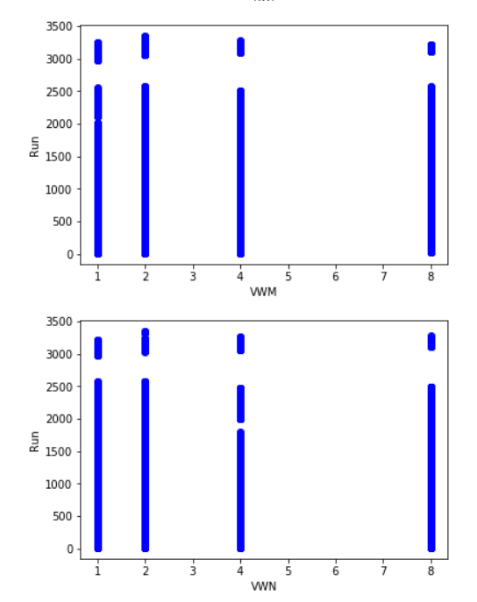
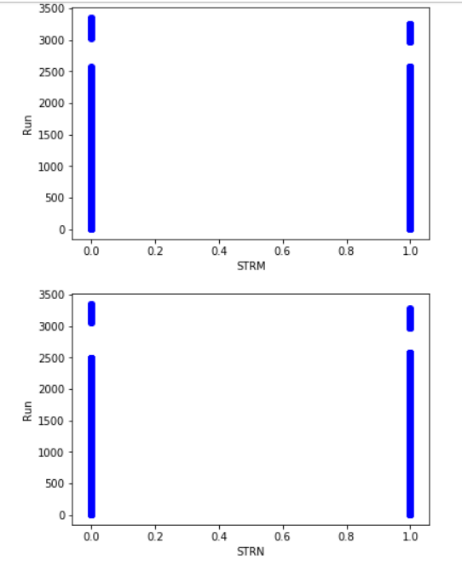
Dependent Variable is Run

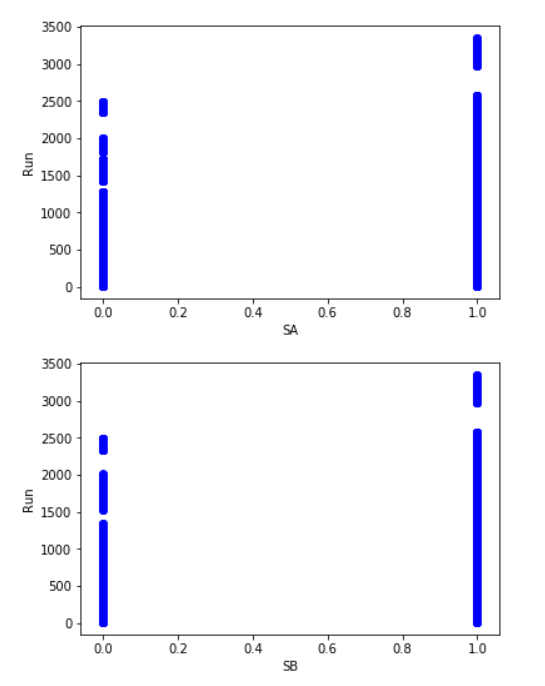
No Missing values in the dataset.

# **Data Preparation and Exploratory Analysis:**

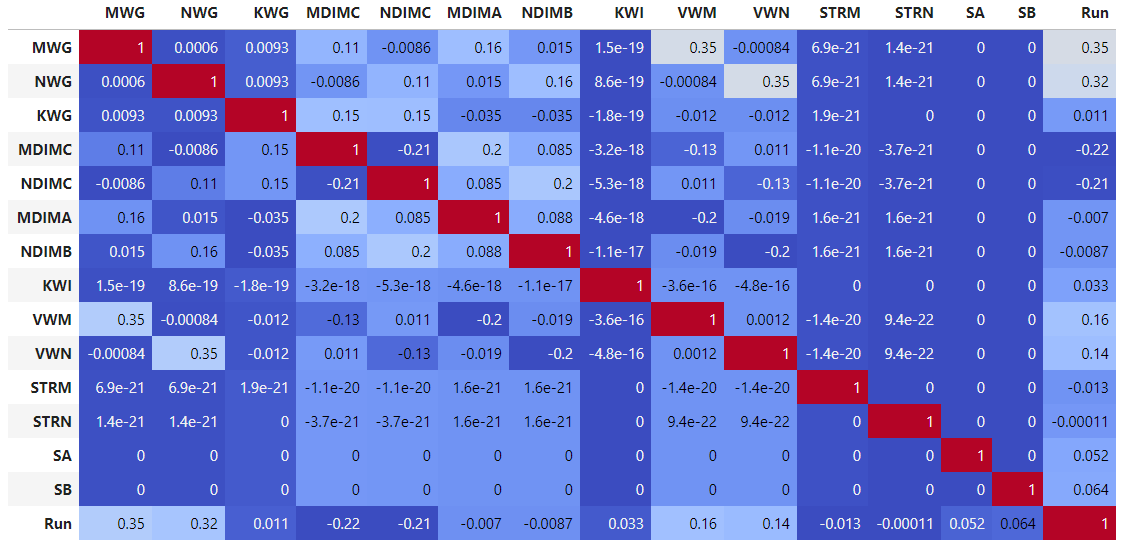
1. Imported the Dataset and analyzed the various parameters to figure out their distribution. Took average of the Run variable as the dependent predictor variable.
2. We have assumed, lower the speed, the processor performs better. Hence, we have coded values below 25th percentile (speed < 0.064621) as 1 and rest as 0.
3. Used the train\_test\_split function to split the data into training (80%) set and test (20%) set.
4. The data is represented in bit format. Each column has values which are multiples of 2. The range of MWG and NWG are in a range between 0 to 128 whereas SA, SB, STRM and STRN are between 0 and 1. Hence, I have performed scaling of the values by subtracting the mean value and dividing by the range.
5. Distribution of various parameters is as follows:



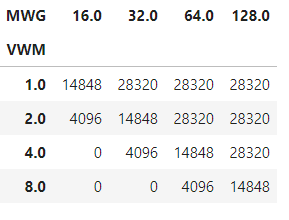


1. Correlation Matrix between the elements is as follows:

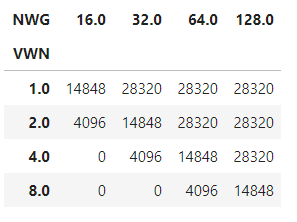


From the correlation matrix, we observe that there is no string correlation between the various parameters, hence no collinearity. The highest colleration value is 0.35 witnessed between pairs (VWM, MWG) and (VWN, NWG).

Hence, we see a crosstabulation between these two pairs.



We see a matrix distribution of variables with a specific pattern. The higher values of MWG and NWG houses higher values of VWM and VWN.



The goal of the model is to classify the average GPU run time as efficient/non-efficient and the various classification algorithms gives us the information on how the function learns from the data. For measuring the performance of the algorithm, I have used ROC curves, classification matrix and area under curve parameters.

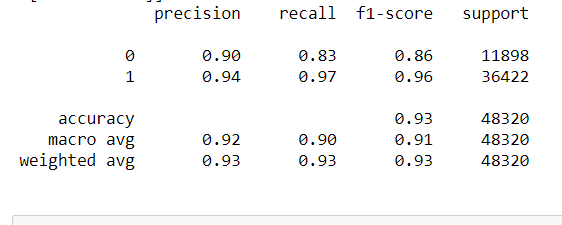
# **Support Vector Machines**

I have used the SVC package from sklearn. It provides with options to change the kernel functions, C and Gamma value. I have performed modelling using 3 kernel functions – linear, RBF (gaussian) and polynomial. The results and interpretation are given below.

**Linear Kernel:**

Implemented the linear kernel package of SVC using the training dataset. Obtained the optimized value for the kernel with value of C as 1. By using the test data, obtained an accuracy of 93.4% for the default case.

The classification report is as below:



We can interpret the precision of the dataset to be 0.93 (weighted average) with an accuracy of 93.4%. Following confusion matrix gives classification rates for this kernel.

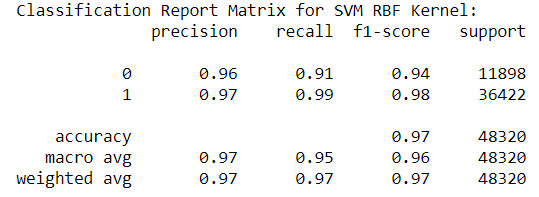


Here we can interpret that the model performs good as we have a high precision and high recall. We will further compare these results with other kernel functions.

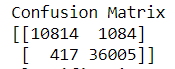
**RBF kernel:**

Implemented the RBF kernel package of SVC using the training dataset. Obtained the optimized value for the kernel with value of C as 1. By using the test data, obtained an accuracy of 96.8% for the default case.

The classification report is as below:

We can interpret the precision of the dataset to be 0.97 (weighted average) with an accuracy of 96.8%. The accuracy of the model is higher than Linear kernel model.

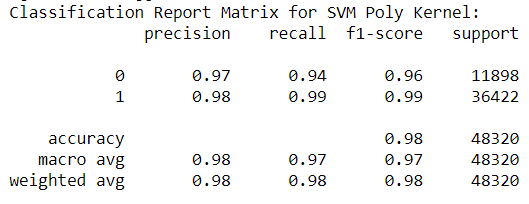
Following confusion matrix gives classification rates for this kernel.

Here we can interpret that the model performs good as we have a high precision and high recall. The precision and recall values are higher than the linear model and hence proves to be a better classifier.

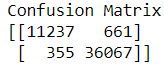
**Polynomial kernel:**

Implemented the polynomial kernel package of SVC using the training dataset. Obtained the optimized value for the kernel with value of C as 1. By using the test data, obtained an accuracy of 97.8% for the default case.

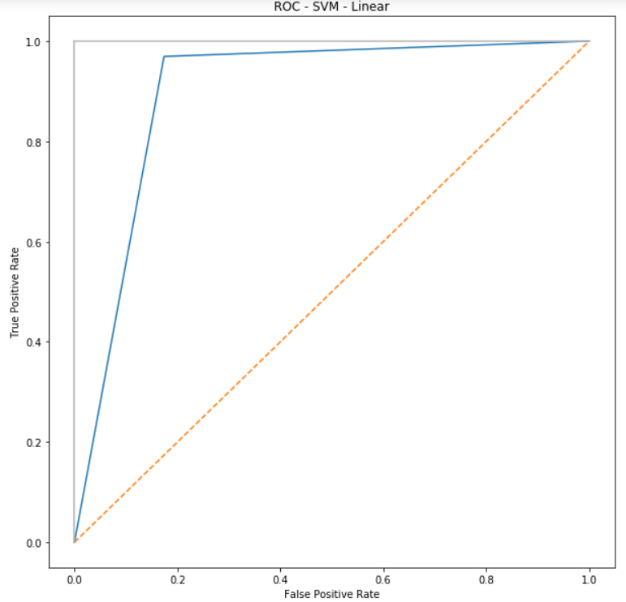
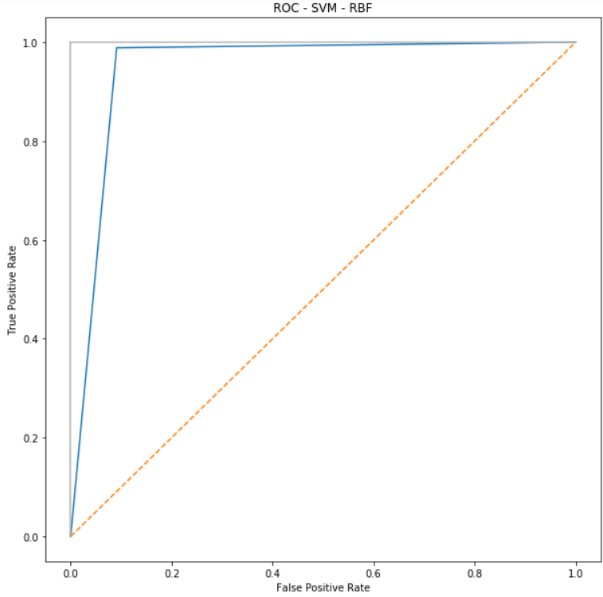
The classification report is as below:

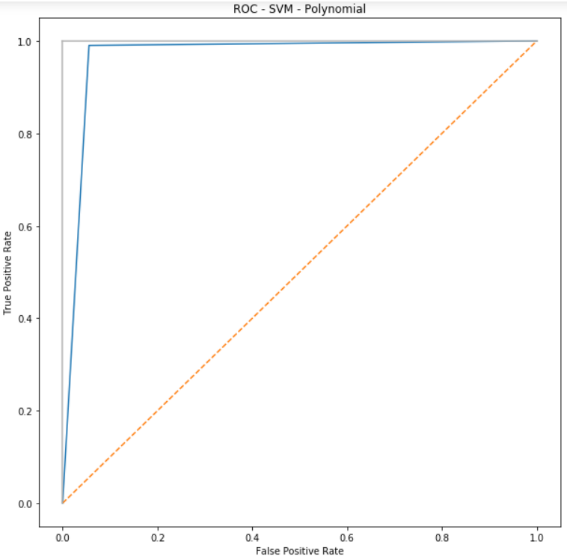
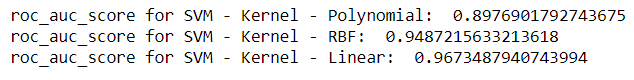
We can interpret the precision of the dataset to be 0.98 (weighted average) with an accuracy of 97.8%. The accuracy of the model is higher than RBF and Linear kernel model.

Following confusion matrix gives classification rates for this kernel.

Here we can interpret that the model performs good as we have a high precision and high recall. The precision and recall values are higher than the linear and RBF model and hence proves to be a better classifier.

From the above reports we can conclude that the polynomial kernel model (degree = 5) performs best among the SVM kernel models. Plotting the ROC Curves:

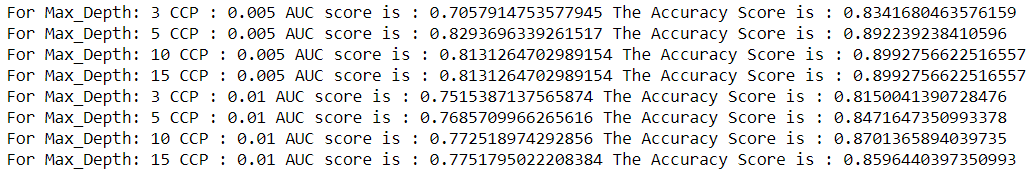


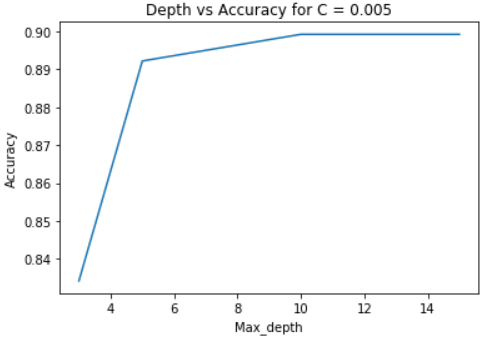
From the above ROC curves, we can interpret that the AUC is highest for Polynomial SVM model. The model performs best with high precision and recall.

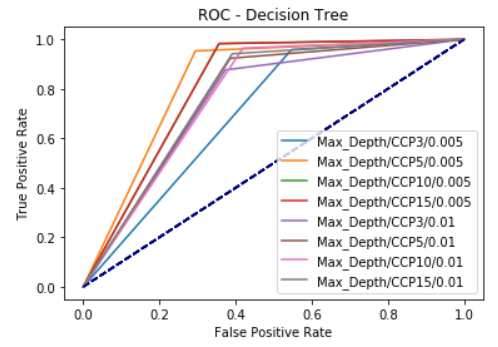
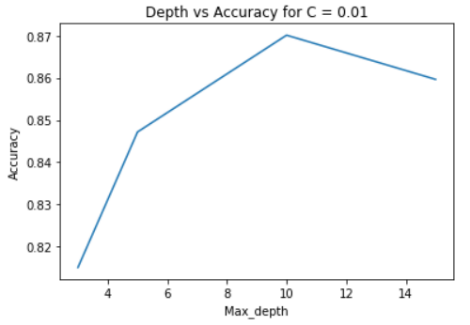
|  |  |  |
| --- | --- | --- |
| **Kernel** | **Accuracy Score%** | **ROC Score %** |
| Linear | 93.4 | 96.7 |
| Gaussian | 97.8 | 94.8 |
| Polynomial | 97.8 | 89.7 |

# **Decision Tree Classifier**

I run the Decision tree classifier initially from the sklearn library and obtain an accuracy of 99.7% on the total tree. I used the gini to calculate the best attribute for each node. As there is not much imbalance in the dataset, we use Gini. I determine the best model by running various combinations for maximum depth and pruning alpha values. Below is the summary statistics:



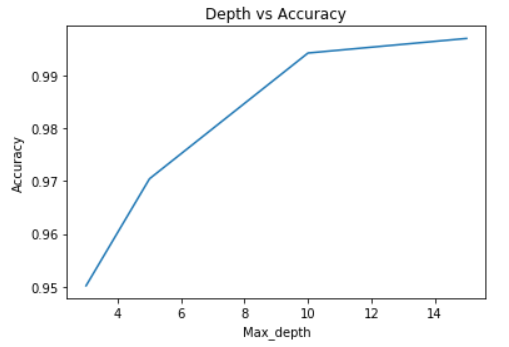
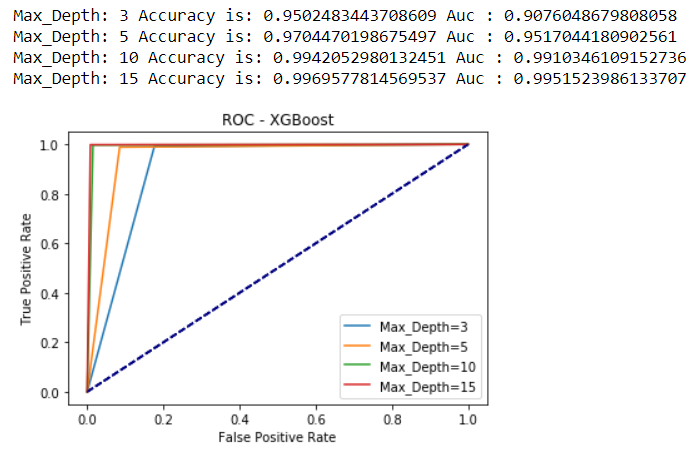
 I can interpret that for depth value of 10 and alpha (CCP) as 0.005, I obtain the highest accuracy and AUC score. Below is the ROC graph –



I can infer from the above graphs for validation of our findings. Accuracy – 89.92% AUC – 0.813

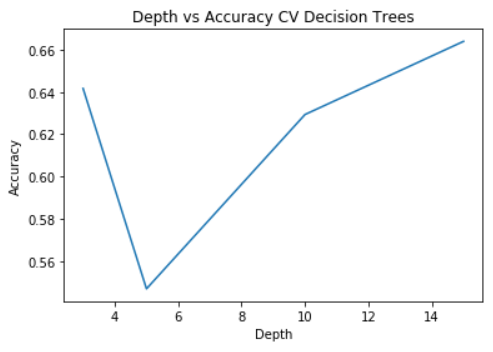
# **XG Boosting**

Implemented the xgboosting package from the sklearn library. I perform the same decision tree algorithm with n\_estimators as 140 for various child weights and gamma value of the tree. I run gridsearchCV to obtain the best parameters for the model with child weight as 1 and gamma as 0. Running this model on various depths and obtaining accuracy scores and ROC curves to obtain the best fit.

Here I noticed that Accuracy and AUC is maximum for depth 10 and 15. As we need a model which is not complex, we can say that model with depth - 10 performs best given the complexity and high accuracy and AUC-0.99. Accuracy – 99.4%

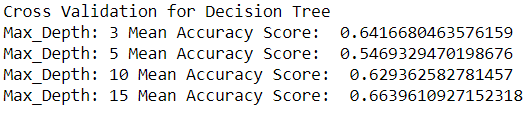
# **Cross Validation**

Performed cross-validation across all the models above with their best parameter values. The results are:

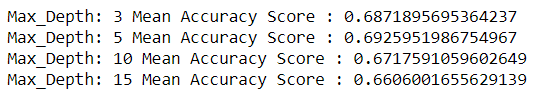
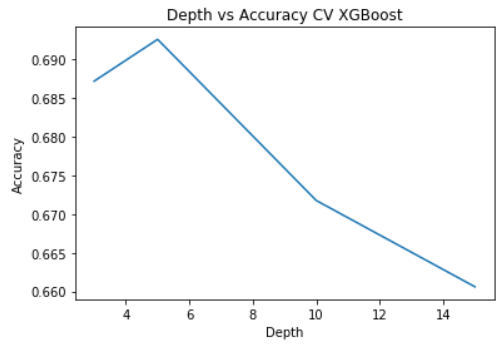
Accuracy of SVM Linear model is 86.35%

Accuracy of SVM rbf is 80.7%

Accuracy of SVM polynomial is 82.1%



Cross validation for XGBoost models are below:



From the above, we obtain varied results from the normal model and the Cross-validation model. As the data set is large – around 2 Lakh data points, running a normal model with random data splitting gives us better results than using cross-validation techniques. From the above cross validation results, we interpret that SVM linear performs best compared to all the other models.

|  |  |
| --- | --- |
| **Kernel** | **Cross Validation Score** |
| linear | 0.86 |
| rbf | 0.80 |
| polynomial | 0.82 |

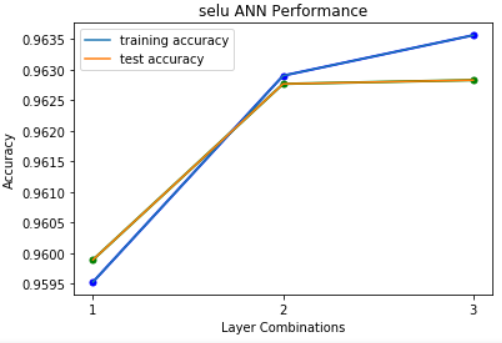
The two best models based on accuracy and roc\_score is the xgboosted model with an accuracy of 99% for a tree depth of 10. The next best model is the SVM polynomial model with an accuracy of 97% and AUC score of 96.7%. Comparing these 2 models, we can imply that the boosted model would be lot faster in the computation time and provides better results compared to all the models. This is because, all our data features are categorical, and the model performs best in this scenario is classifying the output.

# **Neural Networks:**

I have used sequential package from keras to implement the various parameters for neural networks. The dense package is used to add layers and number of nodes into the model. Dense is the layer type that is used which connects all nodes in the previous layer to the nodes in the current layer. I have experimented with number of layers, number of nodes and activation functions and used the best parameters to run batch normalization and dropout networks. To model the nonlinear relationships, the activation functions used here are selu, tanh and relu.

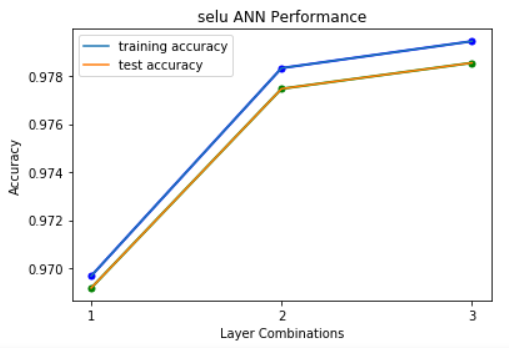
I have used binary cross entropy as the loss model because our target values are in the set {0,1} and accuracy metrics to validate the model. I have used stochastic gradient descent for the model optimizer. Also, I have used early stopping to stop the model if the model stops improving.

Each of the activation functions are run together for 3 combinations of layers and nodes and results obtained are tabulated. The results and interpretation are given below:

**Selu:**

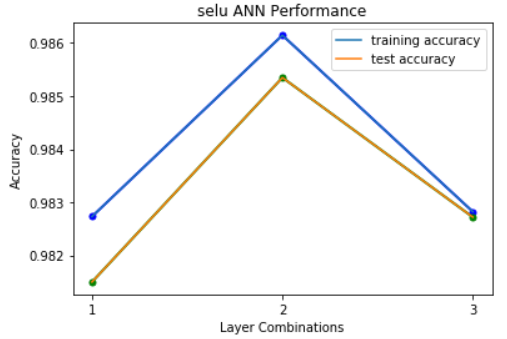
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Combination** | **Nodes** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| 1 | 20 | 95.99% | 0.96 | 0.96 | 0.9365 |
| 2 | 30 | 96.28% | 0.96 | 0.96 | 0.9375 |
| 3 | 40 | 96.28% | 0.96 | 0.96 | 0.9378 |

**1-Layer combination:**

From the above tabulations we can infer that with higher number of nodes, the accuracy and AUC increases but very slightly. We obtain a very high precision and recall score - 0.96. This model performance can further be compared when we increase the layers.

**2-layer combination:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Combination** | **Nodes** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| 1 | 20,20 | 96.92% | 0.97 | 0.97 | 0.9439 |
| 2 | 30,30 | 97.75% | 0.98 | 0.98 | 0.9585 |
| 3 | 40,40 | 97.85% | 0.98 | 0.98 | 0.9614 |

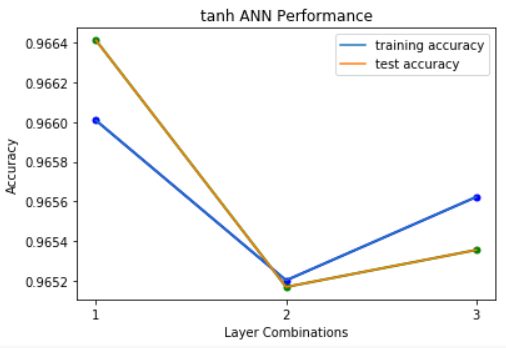
From the above tabulations we can infer that with higher number of nodes, the accuracy and AUC increases but very slightly. We obtain a very high precision and recall score - 0.98. This model performance is better than the model with 1 hidden layer.

**3-layer combination:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Combination** | **Nodes** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| 1 | 20,20,20 | 98.15% | 0.98 | 0.98 | 0.9682 |
| 2 | 30,30,30 | 98.53% | 0.99 | 0.99 | 0.9740 |
| 3 | 40,40,40 | 98.27% | 0.98 | 0.98 | 0.9676 |

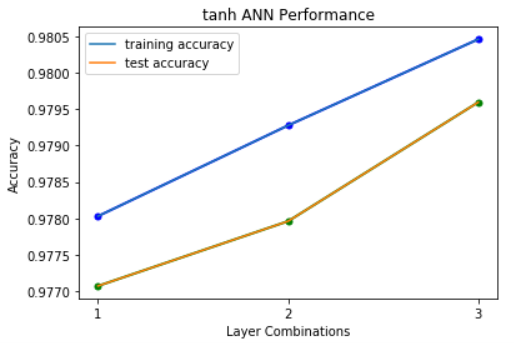
From the above tabulations we can infer that with higher number of nodes, the accuracy and AUC increases but very slightly. We obtain a very high precision and recall score - 0.98. This model performance is better than the model with 1 and 2 hidden layers.

In Selu, we can say that with combination 30,30,30 in the hidden layers, we obtain the highest accuracy of 98.53%. For this combination, we also have a very high precision and recall and AUC of 0.9740. With selu activation, this combination performs best.

**Tanh:**

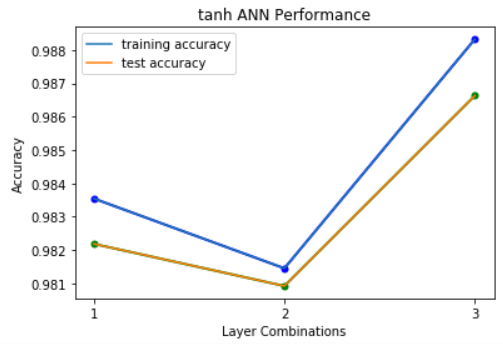
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Combination** | **Nodes** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| 1 | 20 | 96.64% | 0.97 | 0.97 | 0.9478 |
| 2 | 30 | 96.52% | 0.96 | 0.97 | 0.9437 |
| 3 | 40 | 96.54% | 0.97 | 0.97 | 0.9439 |

**1-Layer combination:**

From the above tabulations we can infer that with 20 nodes in hidden layer, we have 96.64% accuracy but reduces to 96.52 when we increase the nodes to 30 and then slightly increases as we increase the nodes to 40. We obtain a very high precision and recall score - 0.97. This model performance can further be compared when we increase the number of layers.

**2-layer combination:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Combination** | **Nodes** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| 1 | 20,20 | 97.71% | 0.98 | 0.98 | 0.9631 |
| 2 | 30,30 | 97.80% | 0.98 | 0.98 | 0.9648 |
| 3 | 40,40 | 97.96% | 0.98 | 0.98 | 0.9666 |

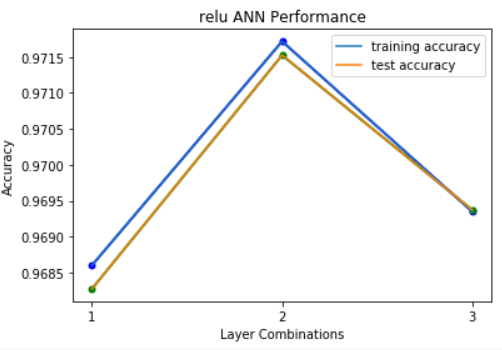
From the above tabulations we can infer that with higher number of nodes, the accuracy and AUC increases but very slightly. We obtain a very high precision and recall score - 0.98. This model performance is better than the model with 1 hidden layer.

**3-layer combination:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Combination** | **Nodes** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| 1 | 20,20,20 | 98.22% | 0.98 | 0.98 | 0.9716 |
| 2 | 30,30,30 | 98.09% | 0.98 | 0.98 | 0.9635 |
| 3 | 40,40,40 | 98.66% | 0.99 | 0.99 | 0.9820 |

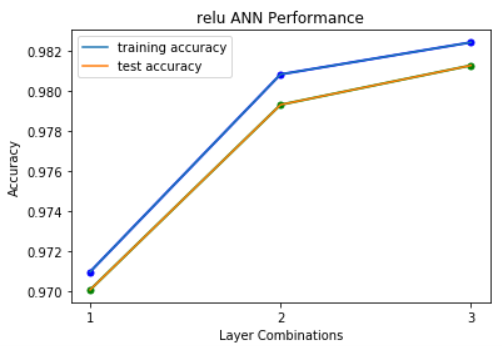
From the above tabulations we can infer that with 20,20,20 nodes in hidden layer, we have 98.22% accuracy but reduces to 98.09% when we increase the nodes to 30,30,30 and then increases as we increase the nodes to 40,40,40. We obtain a very high precision and recall score - 0.99. This model performance is better than the model with 1 hidden layer and 2 hidden layers.

In Tanh, we can say that with combination 40,40,40 in the hidden layers, we obtain the highest accuracy of 98.66%. For this combination, we also have a very high precision and recall and AUC of 0.9820. This combination performs better than selu activation with higher accuracy.

**Relu:**

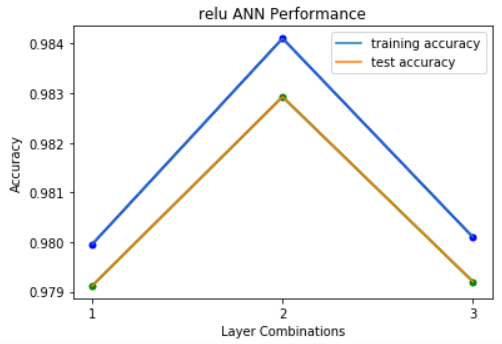
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Combination** | **Nodes** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| 1 | 20 | 96.83% | 0.97 | 0.97 | 0.9496 |
| 2 | 30 | 97.15% | 0.96 | 0.97 | 0.9543 |
| 3 | 40 | 96.94% | 0.97 | 0.97 | 0.9509 |

**1-Layer combination:**

From the above tabulations we can infer that with 20 nodes in hidden layer, we have 96.83% accuracy and increases to 97.15% when we increase the nodes to 30 and then decreases to 96.94% as we increase the nodes to 40. We obtain a very high precision and recall score - 0.97. This model performance can further be compared when we increase the number of layers.

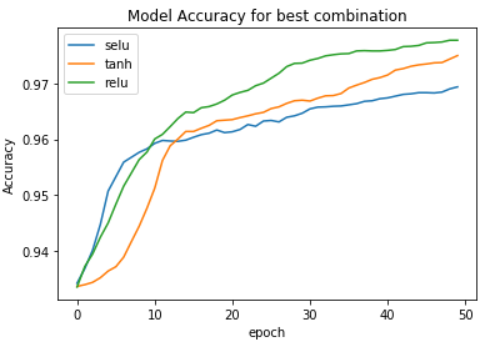
**2-layer combination:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Combination** | **Nodes** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| 1 | 20,20 | 97.01% | 0.97 | 0.97 | 0.9499 |
| 2 | 30,30 | 97.93% | 0.98 | 0.98 | 0.9650 |
| 3 | 40,40 | 98.13% | 0.98 | 0.98 | 0.9673 |

From the above tabulations we can infer that with higher number of nodes, the accuracy and AUC increases but very slightly. We obtain a very high precision and recall score - 0.98. This model performance is better than the model with 1 hidden layer.

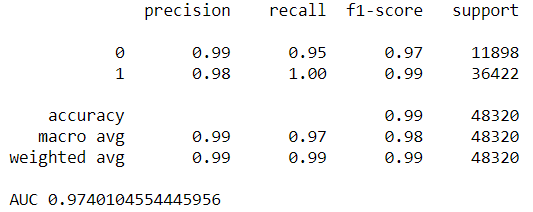
**3-layer combination:**

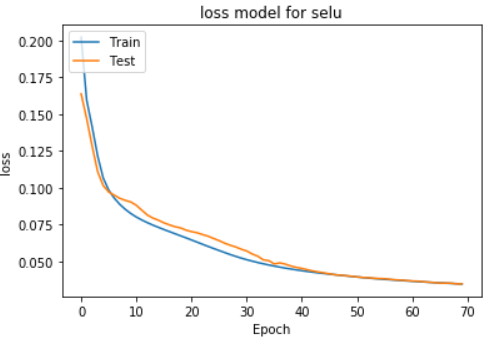
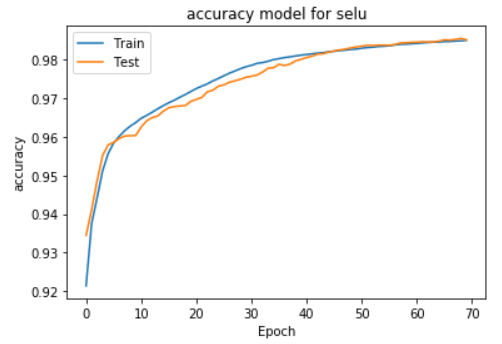
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Combination** | **Nodes** | **Accuracy** | **Precision** | **Recall** | **AUC** |
| 1 | 20,20,20 | 97.91% | 0.98 | 0.98 | 0.9609 |
| 2 | 30,30,30 | 98.29% | 0.98 | 0.98 | 0.9705 |
| 3 | 40,40,40 | 97.92% | 0.99 | 0.99 | 0.9601 |

From the above tabulations we can infer that with 20,20,20 nodes in hidden layer, we have 98.22% accuracy but reduces to 98.09% when we increase the nodes to 30,30,30 and then increases as we increase the nodes to 40,40,40. We obtain a very high precision and recall score - 0.99. This model performance is better than the model with 1 hidden layer and 2 hidden layers.

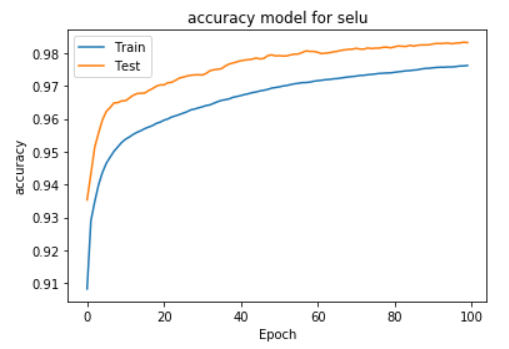
In Relu, we can say that with combination 40,40,40 in the hidden layers, we obtain the highest accuracy of 98.29%. For this combination, we have a very high precision and recall and AUC of 0.9705. But, comparing this combination with selu and tanh activation, accuracy metric is slightly lesser.

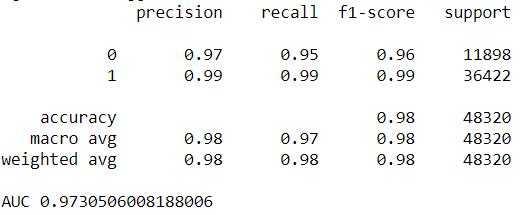
We can see that for combination with 40,40,40 in hidden layers and tanh activation, the model accuracy is highest – 98.66% but considering the complexity, we choose the model with 30,30,30 and selu activation where the model accuracy is 98.53%. This model is finalized as we can compromise for model complexity with a better accuracy.

For the model combination with 30,30,30 with selu activation, we obtain the classification report as follows:



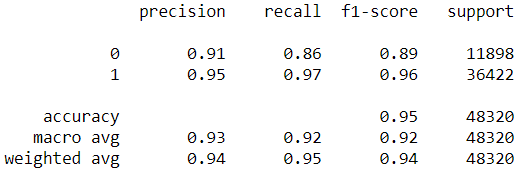
**Batch Normalization:**

We implement batch normalization to our model and observe the findings. Batch normalization reduces the covariance shift – the amount by which the hidden unit values shift around. It performs normalization to every hidden layer. I have used the BatchNormalization() package from keras. By fitting the model, the classification report obtained is as follows:

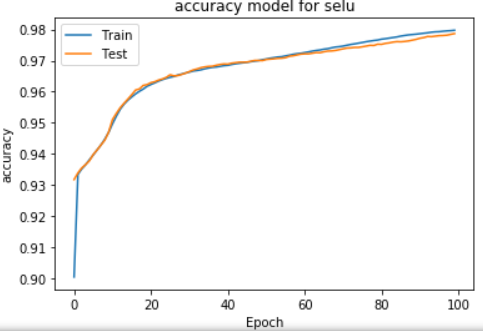


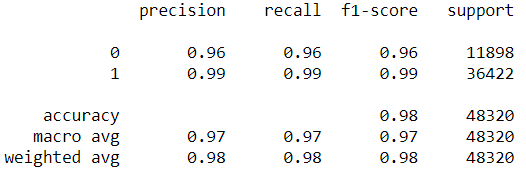
**Dropout:**

We implement dropout to our model and observe the findings for inference. Dropout removes the dependency of nodes within each other. By using dropout, we ignore a few nodes during each stage, done by partial learning. I have used the Dropout() package from keras. By fitting the model, the classification report obtained is as follows:



**Adam Optimizer:**

I have experimented with the adam optimizer for model optimization. Adam optimizer tends to converge faster and hence provides quicker results. By fitting the model, the classification report obtained is as follows:

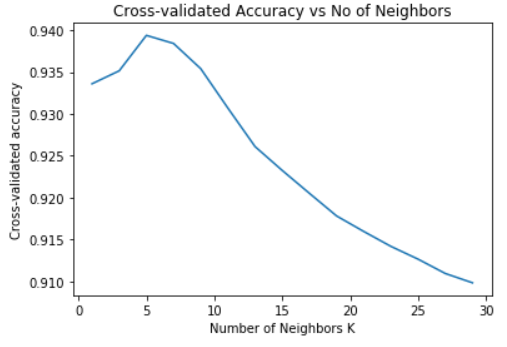
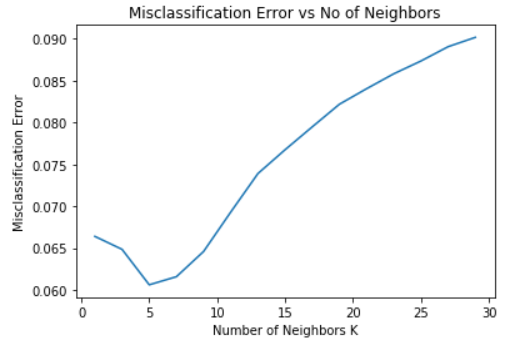


Here you can see that the accuracy is less than the gradient descent optimizer as this model has a tradeoff for faster performance with accuracy. Hence, if the model requires quick training, we tend to use adam optimizer.

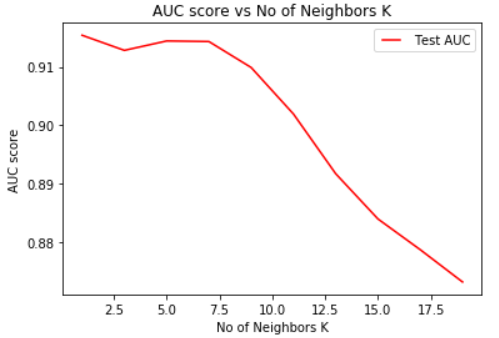
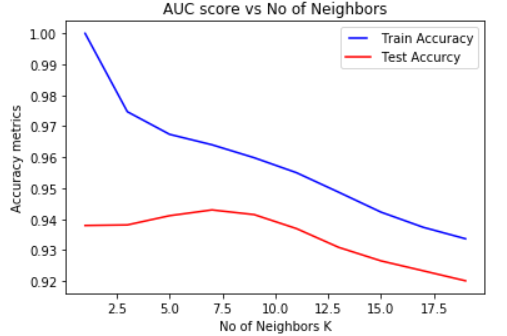
Based on all the models, gradient descent optimizer, batch normalization, dropout and adam optimizer, the best model in neural network is with nodes 30,30,30 with selu activation.

# **KNN model**

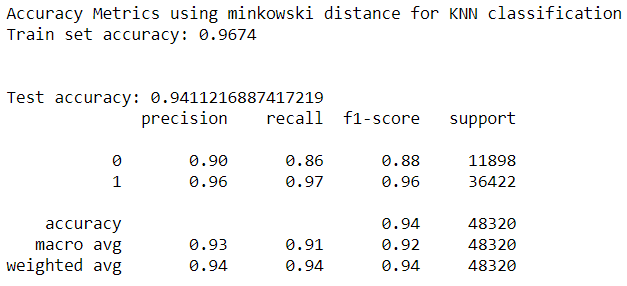
I have used the KNeighborsCLassifier package from sklearn. It provides with parameters to change the neighbors and distance metrics. I have performed modelling by varying the number of neighbors to find the optimal value. The various distances that have been experimented with are minkowski, Euclidean, manhattan and Chebyshev. I have used gridsearchCV to find the best set of parameters for the KNN algorithm.

Experimenting with the number of neighbors, the results are stated below:

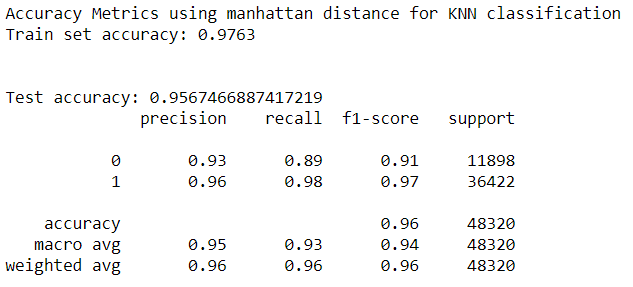
`

We see from the above graphs that optimal number of neighbors is 5. From the accuracy curve we see that for n = 5, the cross-validated accuracy is the highest at 93.8% and the misclassification error is the least at 6.2%.

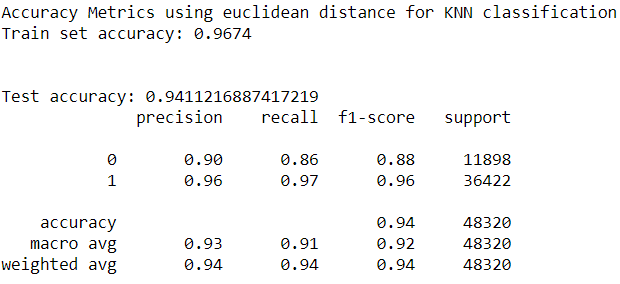
From the above curves, we can see that the best combination of higher AUC score and Accuracy measure is found to be for number of neighbors as 5, observing the training set accuracy to be 96.8% and test set accuracy to be 94.2%.

By varying the various distance metrics, we obtain the classification report and accuracy measures as follows.

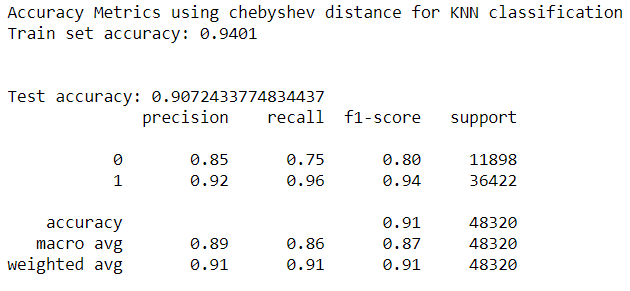
We can interpret the precision of the dataset to be 0.94 (weighted average) using minkowski distance with an accuracy of 94.1% for the test data.



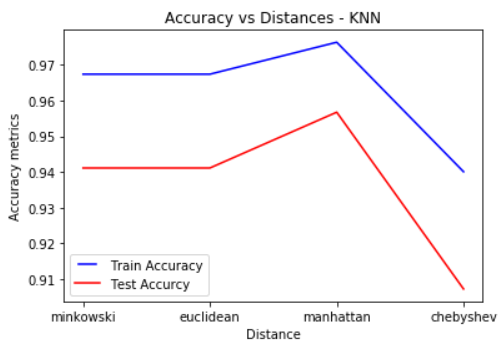
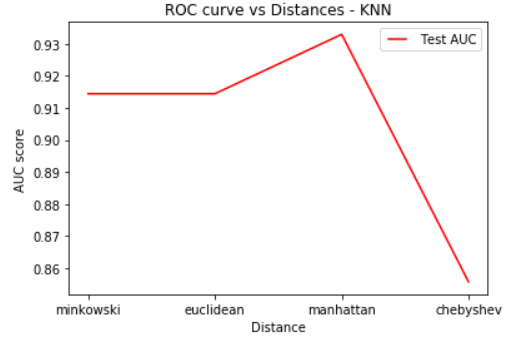
We can interpret the precision of the dataset to be 0.96 (weighted average) using mannattan distance with an accuracy of 95.6% for the test data. The model performs better than minkowski distance metricusing Manhattan distance parameter.



We can interpret the precision of the dataset to be 0.94 (weighted average) using Euclidean with an accuracy of 94.1% for the test data. The model performs similar to the model using Minkowski distance parameters but has less performance compared to manhattan distance metrics.

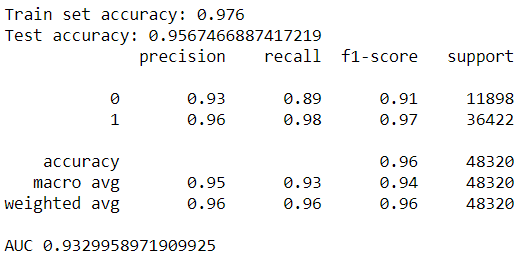
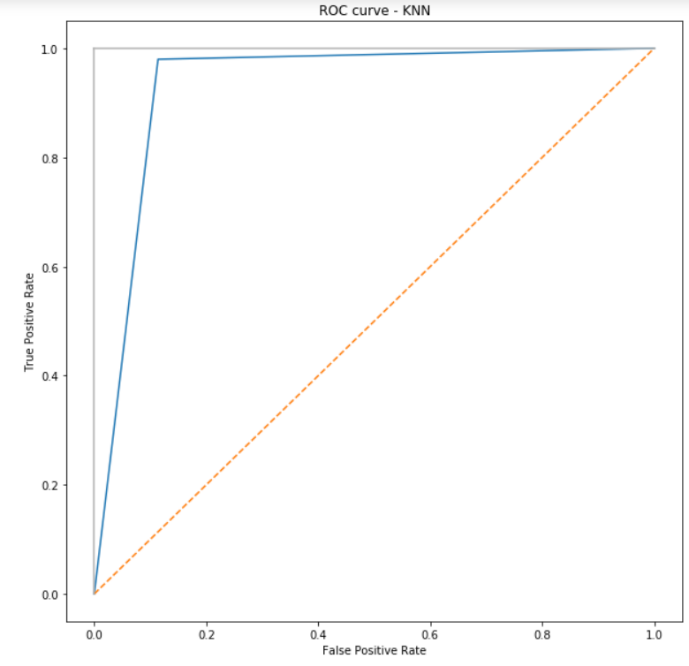


We can interpret the precision of the dataset to be 0.91 (weighted average) using Chebyshev with an accuracy of 90.7% for the test data. The model has less performance compared to manhattan, Euclidean and minkowski distance metrics.

Below we plot the ROC and accuracy curves for comparison between the different distance metrics

Here we can conclude that for this data, the model performs good when we use the manhattan distance parameter. We obtain the highest accuracy and AUC score for the same.

We perform grid search CV to validate our findings with multiple values for neighbor and distance parameters. Through this we can ascertain our findings. We obtain the best possible combination with number of neighbors as 5 and distance metric as manhattan. We then create a model using the best parameters obtained. The classification report for the best model is:



The AUC for the best subset curve is 0.932. From the classification report, this model has a training accuracy of 97.6% and test accuracy of 95.67% with a precision of 0.96. This model performs best with high precision, high recall.

# **Comparison between models:**

For Neural Networks, the best model obtained is 30,30,30 with selu activation and accuracy of 98.53%. For KNN, the best model obtained is with 5 neighbors and manhattan distance metric with an accuracy of 95.67%. The best model for this data would be the Neural Network Model.

Comparing with all the models, we see that xgBoost model performs the best.

|  |  |  |
| --- | --- | --- |
| Model | Accuracy | Rank |
| SVM | 97% | 3 |
| Decision Tree | 89.92% | 5 |
| XGBoost | 99% | 1 |
| Neural Networks | 98.53% | 2 |
| KNN | 95.67% | 4 |