**Artificial Neural Networks (ANN) and K Nearest Neighbors (KNN)**

**AMIT V GOTTIPATI**

## Introduction

As part of this binary classification project Artificial Neural Networks (ANN) and K Nearest Neighbors were implemented in Python and Google colab using the SGEMM GPU kernel performance data set from UCI Machine Learning repository -> Dataset and on the Rain in Australia dataset from Kaggle -> Dataset2 . The project involved data pre-processing, data visualization, algorithm application, prediction and experimenting with different parameters like Number of hidden layers, Neurons, Activation Functions, Optimizer, Batch size, epoch, K, etc.

## Dataset Description

### Dataset1 - SGEMM GPU :

- The First dataset contains 241600 observations and 18 variables with no missing values.
- There are 18 variables, the first 14 are parameters upon which we will train our model. The first 10 parameters are ordinal with 4 levels, the next 4 are binary and last 4 variables are 4 different run times for the SGEMM GPU
- The description for the independent variables is given in the dataset link.
- The dataset was divided into train and test with a 70:30 split.
- For dependent variable 'avg' was created using average of the 4 run times provided which tells us the average running time for matrix-matrix product of the four runs. For classification the dependent variable 'avg' was converted into a binary variable using median as the threshold. (High/Low)

### Dataset2:

- This Dataset contains 142193 observations with 24 variables
- Here the Target variable is Rain Tomorrow, we need to predict if it will rain tomorrow or not.
- It is given that Risk_MM variable is a very good predictor of the outcome and is dropped.
- Variables with more than 35% missing values were dropped and Last observation carried forward LOCF was used to impute the remaining missing values

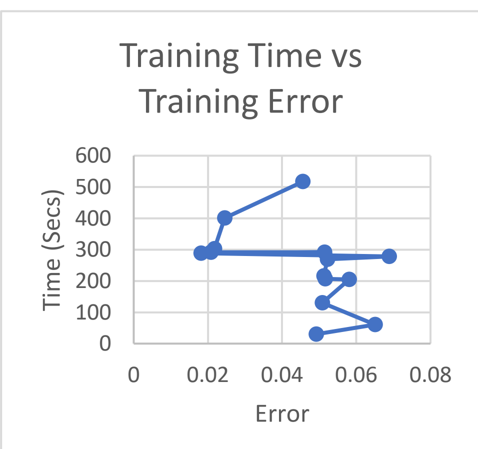
**Artificial Neural Networks**

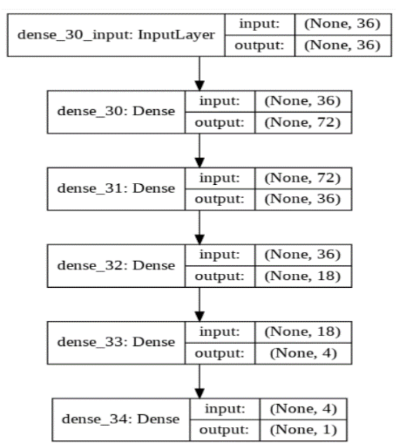
**Dataset-1:**

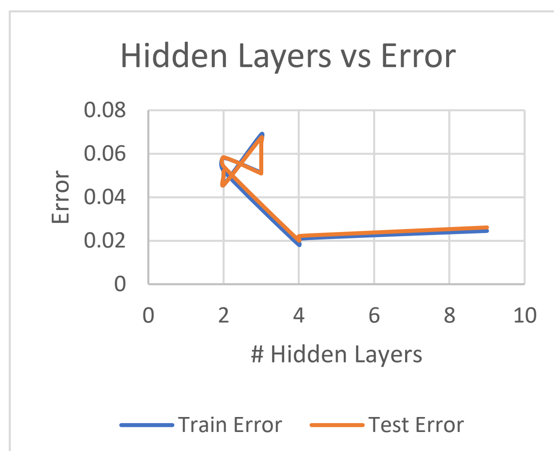
| Tanh = T | Sigmoid = S | Relu = R | Adam = A | SGD =SG | SoftMax = SX |
|---|---|---|---|---|---|

**Dataset1**

| Model # | No of Hidden Layers | Neurons in each Hidden layer | Activation Function in Each Layer (Hidden & Output Layer) | Optimizer | Batch Size | Epoch | Loss | Train Error | Test Error | Training Time |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4,4 | R,R,S | A | 10 | 5 | 0.1221 | 0.05162 | 0.052 | 207 |
| 2 | 2 | 4,4 | R,R,S | A | 20 | 5 | 0.1231 | 0.0508 | 0.0515 | 130 |
| 3 | 2 | 4,4 | R,R,S | A | 100 | 5 | 0.116 | 0.0492 | 0.0498 | 30 |
| 4 | 2 | 4,4 | R,R,S | A | 10 | 10 | 0.1094 | 0.0456 | 0.0456 | 517 |
| 5 | 4 | 4,4,4,4 | R,R,R,R,S | A | 10 | 5 | 0.6934 | 0.4999 | 0.5005 | 318 |
| 6 | 4 | 4,4,4,4 | R,R,R,R,S | A | 100 | 10 | 0.6931 | 0.4999 | 0.5005 | 70 |
| 7 | 3 | 4,4,4 | R,R,R,S | A | 10 | 5 | 0.1838 | 0.0689 | 0.0674 | 278 |
| 8 | 3 | 4,4,4 | R,R,R,S | A | 50 | 5 | 0.174 | 0.0651 | 0.0647 | 61 |
| 9 | 3 | 4,4,4 | R,R,T,S | A | 10 | 5 | 0.1219 | 0.0514 | 0.0509 | 292 |
| 10 | 3 | 4,4,4 | R,R,T,S | SG | 10 | 5 | 0.1242 | 0.0513 | 0.0509 | 216 |
| 11 | 2 | 4,2 | R,T,S | SG | 10 | 5 | 0.1397 | 0.0581 | 0.0584 | 205 |
| 12 | 2 | 4,2 | R,T,S | A | 10 | 5 | 0.1241 | 0.0522 | 0.0542 | 269 |
| 13 | 4 | 72,36,18,4 | R,T,T,T,S | A | 10 | 5 | 0.0469 | 0.0181 | 0.0201 | 289 |
| 14 | 4 | 72,36,18,4 | R,T,R,T,S | A | 10 | 5 | 0.0509 | 0.0208 | 0.0221 | 292 |
| 15 | 5 | 36,36,18,18,4,1 | R,T,S,S,T,S | A | 10 | 5 | 0.0539 | 0.0218 | 0.023 | 303 |
| 16 | 9 | 36,32,28,24,20,16,12,8,4 | R,T,T,T,T,T,S,S,T,S | A | 10 | 5 | 0.0602 | 0.0245 | 0.0261 | 401 |



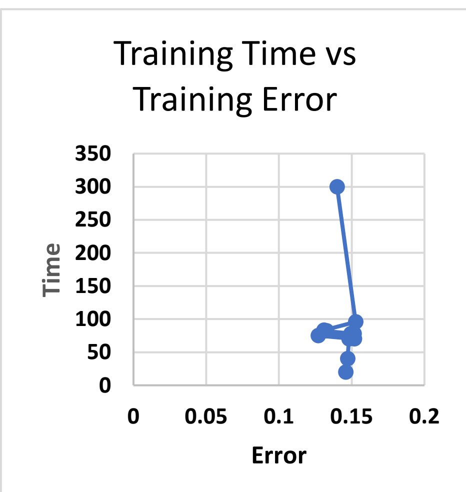



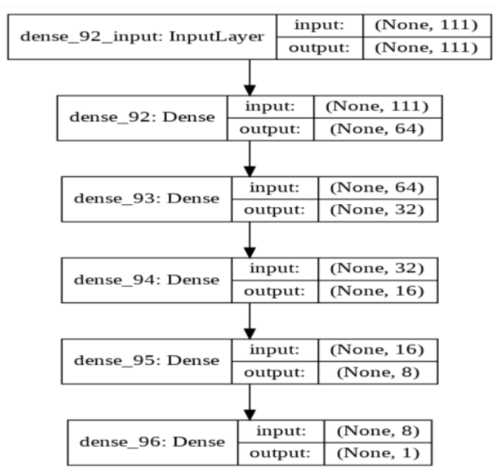



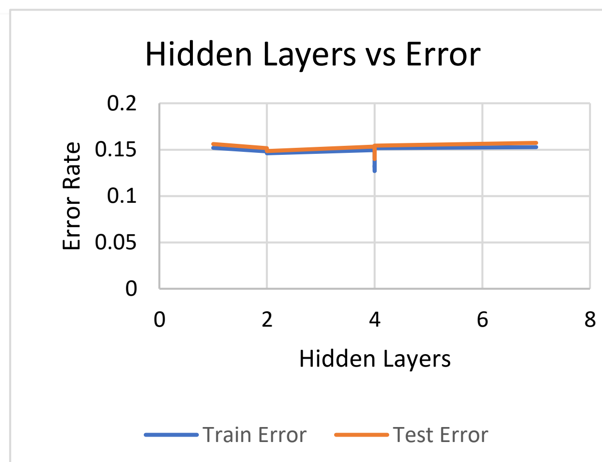

From this experiment we saw that Adam (RMSProp + Momentum) and SGD(Stochastic Gradient Descent) optimizer gave similar results. We got the best result when we used a 4 hidden layer neural network with Relu and Tanh activation functions with decreasing number of neurons. We also observed lowest loss for this result. Increasing the complexity of neural network by increasing number of hidden layers decrease the train and test error but it started increasing again with high number of hidden layers .

**Dataset-2:**

| Dataset2 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Model # | No of Hidden Layers | Neurons in each Hidden layer | Activation Function in Each Layer (Hidden & Output Layer) | Optimizer | Batch Size | Epoch | Loss | Train Error | Test Error | Training Time |
| 1 | 2 | 4,4 | R,R,S | A | 10 | 5 | 0.3455 | 0.1482 | 0.1516 | 70 |
| 2 | 2 | 4,4 | R,R,S | A | 20 | 5 | 0.3426 | 0.1473 | 0.1489 | 40 |
| 3 | 2 | 4,4 | R,R,S | A | 100 | 5 | 0.3398 | 0.146 | 0.1484 | 20 |
| 4 | 4 | 4,4,4,4 | R,R,R,R,S | A | 10 | 5 | 0.3526 | 0.1495 | 0.1532 | 78 |
| 5 | 4 | 64,32,16,8 | R,R,R,R,S | A | 10 | 5 | 0.3092 | 0.1309 | 0.1429 | 83 |
| 6 | 4 | 64,32,16,8 | R,T,T,R,S | A | 10 | 5 | 0.3064 | 0.1308 | 0.1418 | 82 |
| 7 | 4 | 64,32,16,8 | R,S,S,R,S | A | 10 | 5 | 0.3102 | 0.1328 | 0.1409 | 82 |
| 8 | 4 | 64,32,16,8 | R,S,S,R,S | SG | 10 | 5 | 0.2983 | 0.127 | 0.1399 | 75 |
| 9 | 4 | 64,32,16,8 | R,S,S,R,SX | SG | 10 | 20 | 11.84 | 0.7771 | 0.7731 | 320 |
| 10 | 4 | 64,32,16,8 | T,T,T,T,S | SG | 10 | 20 | 0.3309 | 0.14 | 0.1455 | 300 |
| 11 | 4 | 64,64,64,64 | T,T,T,T,S | SG | 10 | 5 | 0.3581 | 0.1517 | 0.1544 | 78 |
| 12 | 1 | 64 | T,S | SG | 10 | 5 | 0.3571 | 0.1519 | 0.1562 | 70 |
| 13 | 11 | 100,90,80,70,60,50,40,30,20,10,5 | T,T,T,T,T,S,S,T,T,T,T,S | SG | 10 | 5 | 0.5307 | 0.223 | 0.223 | 95 |
| 14 | 7 | 200,100,100,100,75,50,25 | T,T,T,T,T,T,T,S | SG | 10 | 5 | 0.3617 | 0.1527 | 0.1573 | 96 |




From this experiment we got the best result when we used a 4 Hidden layer neural network with decreasing number of neurons, Relu and Sigmoid Activation functions and SG optimizer. The Training and Testing Error remains very similar for different neural networks.

## K Nearest Neighbors (KNN)

We have used PyCaret library in python to implement KNN. For initial models we have used Minkowski metric for Dataset1 and Dataset2. Minkowski distance is a generalization of both the Euclidean distance and the Manhattan distance and hence can be a good fit for our model. Using Pycaret we have created tuned and calibrated models for KNN. We have used a 2 fold cross validation for all the models.

### Dataset-1:

Train and Test Errors Respectively

```
print(knn)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
print(tuned_knn)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='manhattan',
                     metric_params=None, n_jobs=None, n_neighbors=14, p=2,
                     weights='distance')
```

```
print(calibrated_knn)
```

```
CalibratedClassifierCV(base_estimator=KNeighborsClassifier(algorithm='auto',
                                                           leaf_size=30,
                                                           metric='minkowski',
                                                           metric_params=None,
                                                           n_jobs=None,
                                                           n_neighbors=5, p=2,
                                                           weights='uniform'),
                       cv=2, method='sigmoid')
```

```
print(calibrated_knn_isotonic)
```

```
CalibratedClassifierCV(base_estimator=KNeighborsClassifier(algorithm='auto',
                                                           leaf_size=30,
                                                           metric='minkowski',
                                                           metric_params=None,
                                                           n_jobs=None,
                                                           n_neighbors=5, p=2,
                                                           weights='uniform'),
                       cv=2, method='isotonic')
```

1)KNN

| | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|
| 0 | 0.9499 | 0.9863 | 0.9381 | 0.9608 | 0.9493 | 0.8998 |
| 1 | 0.9494 | 0.9862 | 0.9367 | 0.9612 | 0.9488 | 0.8989 |
| Mean | 0.9497 | 0.9862 | 0.9374 | 0.9610 | 0.9490 | 0.8994 |
| SD | 0.0002 | 0.0001 | 0.0007 | 0.0002 | 0.0003 | 0.0005 |

| | Model | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|---|
| 0 | K Neighbors Classifier | 0.9655 | 0.9932 | 0.9517 | 0.9787 | 0.965 | 0.931 |

2)Tuned KNN

| | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|
| 0 | 0.9594 | 0.9935 | 0.9401 | 0.9779 | 0.9586 | 0.9188 |
| 1 | 0.9576 | 0.9931 | 0.9388 | 0.9754 | 0.9568 | 0.9152 |
| Mean | 0.9585 | 0.9933 | 0.9394 | 0.9766 | 0.9577 | 0.9170 |
| SD | 0.0009 | 0.0002 | 0.0006 | 0.0012 | 0.0009 | 0.0018 |

| | Model | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|---|
| 0 | K Neighbors Classifier | 0.969 | 0.9965 | 0.9495 | 0.9879 | 0.9683 | 0.9379 |

3)Calibrated KNN (Platt sigmoid)

| | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|
| 0 | 0.9551 | 0.9905 | 0.9552 | 0.9550 | 0.9551 | 0.9102 |
| 1 | 0.9529 | 0.9903 | 0.9539 | 0.9519 | 0.9529 | 0.9057 |
| Mean | 0.9540 | 0.9904 | 0.9546 | 0.9535 | 0.9540 | 0.9080 |
| SD | 0.0011 | 0.0001 | 0.0006 | 0.0015 | 0.0011 | 0.0022 |

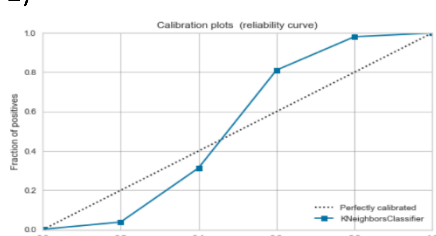
| | Model | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|---|
| 0 | Calibrated Classifier C V | 0.9651 | 0.9943 | 0.9633 | 0.9668 | 0.965 | 0.9302 |

4) Calibrated Isotonic KNN

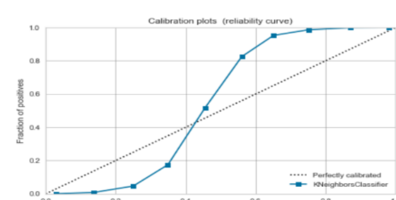
| | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|
| 0 | 0.9511 | 0.9905 | 0.9254 | 0.9756 | 0.9498 | 0.9023 |
| 1 | 0.9500 | 0.9904 | 0.9239 | 0.9747 | 0.9486 | 0.8999 |
| Mean | 0.9505 | 0.9904 | 0.9247 | 0.9751 | 0.9492 | 0.9011 |
| SD | 0.0006 | 0.0001 | 0.0008 | 0.0004 | 0.0006 | 0.0012 |

| | Model | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|---|
| 0 | Calibrated Classifier C V | 0.9651 | 0.9943 | 0.9633 | 0.9668 | 0.965 | 0.9302 |

1)

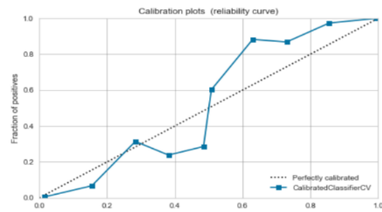



2)

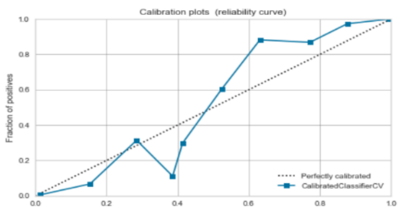


3)

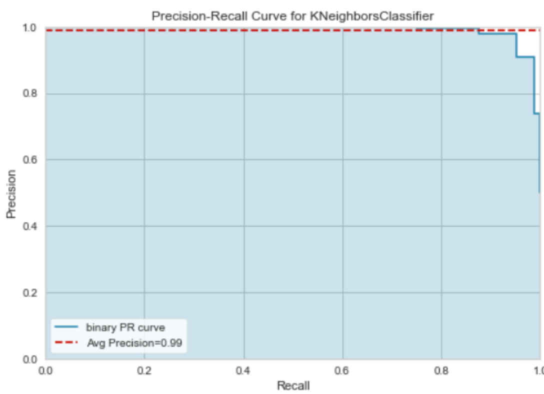
4)

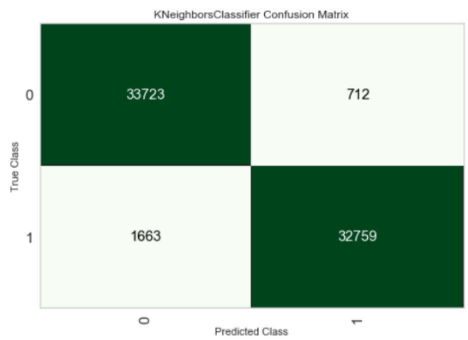


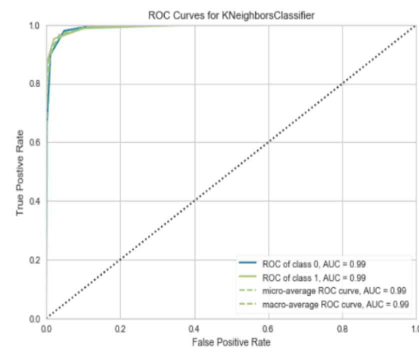

We observe from this experiment that Tuned KNN gave us the best results with K=14 and manhattan as the distance metric. We also see that calibration plots for KNN and Tuned KNN are s curved. We tried to calibrate them using Platt sigmoid curve and isotonic regression and gave similar results.



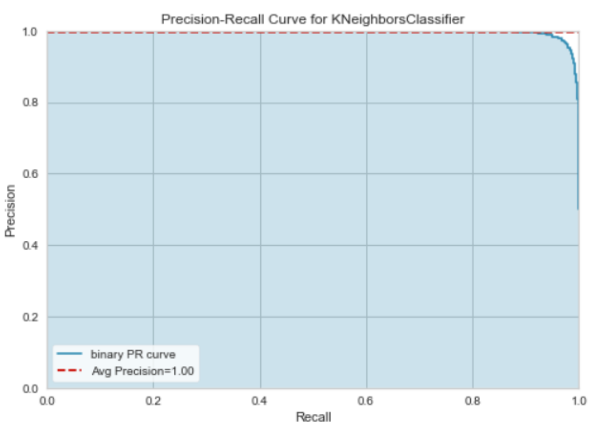

1)KNN

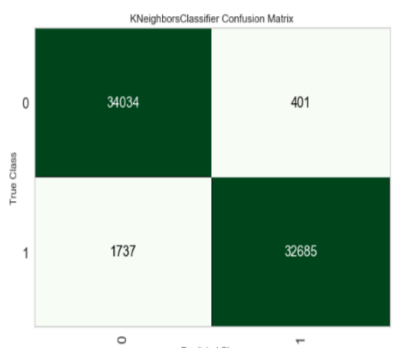


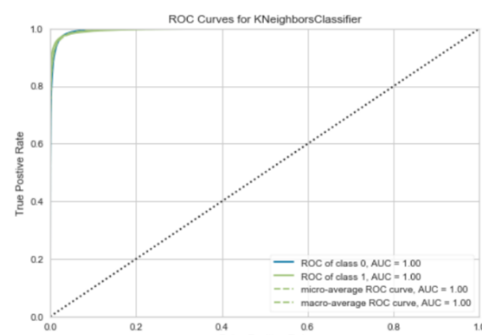




2)Tuned KNN



We ran all model comparison to see which model performed best. We can see that CatBoostClassifier gives best training accuracy. We need to apply on test data to see if overfitting is not taking place. A 2 fold cross validation has been applied to all models.

| | Model | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|---|
| 0 | CatBoost Classifier | 0.9839 | 0.999 | 0.9785 | 0.9893 | 0.9838 | 0.9678 |
| 1 | Decision Tree Classifier | 0.9834 | 0.9834 | 0.9841 | 0.9827 | 0.9834 | 0.9668 |
| 2 | Extra Trees Classifier | 0.9823 | 0.9984 | 0.9757 | 0.9888 | 0.9822 | 0.9646 |
| 3 | Random Forest Classifier | 0.9722 | 0.996 | 0.9571 | 0.9869 | 0.9718 | 0.9444 |
| 4 | Light Gradient Boosting Machine | 0.9672 | 0.996 | 0.9485 | 0.9853 | 0.9666 | 0.9344 |
| 5 | K Neighbors Classifier | 0.9497 | 0.9862 | 0.9374 | 0.961 | 0.949 | 0.8994 |
| 6 | Gradient Boosting Classifier | 0.9122 | 0.9614 | 0.8919 | 0.9297 | 0.9104 | 0.8244 |
| 7 | Extreme Gradient Boosting | 0.908 | 0.9604 | 0.8895 | 0.9236 | 0.9062 | 0.816 |
| 8 | SVM - Linear Kernel | 0.839 | 0 | 0.8359 | 0.841 | 0.8385 | 0.6779 |
| 9 | Logistic Regression | 0.8227 | 0.8842 | 0.8211 | 0.8237 | 0.8224 | 0.6454 |
| 10 | Ridge Classifier | 0.8226 | 0 | 0.8155 | 0.8271 | 0.8213 | 0.6451 |
| 11 | Linear Discriminant Analysis | 0.8226 | 0.8832 | 0.8156 | 0.8271 | 0.8213 | 0.6451 |
| 12 | Ada Boost Classifier | 0.8005 | 0.8765 | 0.8202 | 0.7891 | 0.8044 | 0.6011 |
| 13 | Naive Bayes | 0.7876 | 0.8244 | 0.7371 | 0.8198 | 0.7763 | 0.5752 |
| 14 | Quadratic Discriminant Analysis | 0.6809 | 0.7299 | 0.711 | 0.6706 | 0.6902 | 0.3617 |

## Dataset-2:

### 1)KNN

| | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|
| 0 | 0.8352 | 0.8181 | 0.4565 | 0.7053 | 0.5543 | 0.4588 |
| 1 | 0.8331 | 0.8175 | 0.4482 | 0.7006 | 0.5467 | 0.4504 |
| Mean | 0.8342 | 0.8178 | 0.4523 | 0.7030 | 0.5505 | 0.4546 |
| SD | 0.0010 | 0.0003 | 0.0041 | 0.0024 | 0.0038 | 0.0042 |

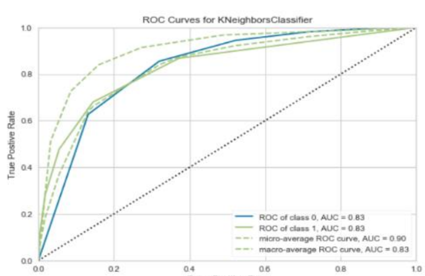
```
1  print(knn)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

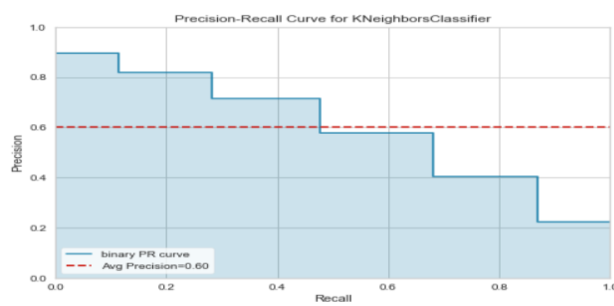
```
1  print(tuned_knn)
```

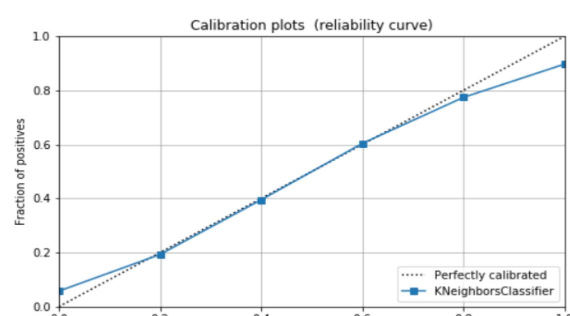
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                     weights='distance')
```

| | Model | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|---|
| 0 | K Neighbors Classifier | 0.84 | 0.8314 | 0.478 | 0.7149 | 0.573 | 0.4793 |

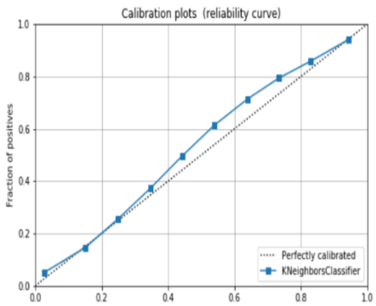


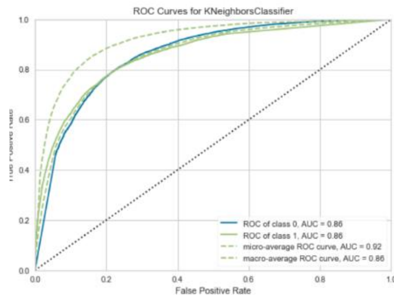




2) Tuned KNN

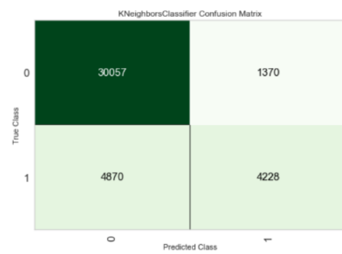


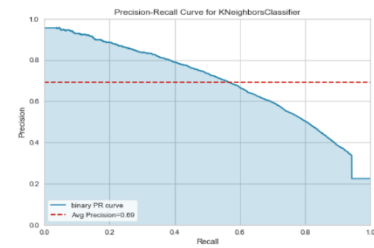




| | Model | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|---|
| 0 | CatBoost Classifier | 0.8573 | 0.8848 | 0.5271 | 0.764 | 0.6238 | 0.5394 |
| 1 | Light Gradient Boosting Machine | 0.853 | 0.8768 | 0.5168 | 0.7509 | 0.6122 | 0.5253 |
| 2 | Extra Trees Classifier | 0.846 | 0.8709 | 0.4256 | 0.7922 | 0.5537 | 0.4707 |
| 3 | Gradient Boosting Classifier | 0.8459 | 0.8612 | 0.4782 | 0.7443 | 0.5822 | 0.4932 |
| 4 | Extreme Gradient Boosting | 0.8459 | 0.861 | 0.4681 | 0.7515 | 0.5769 | 0.4888 |
| 5 | Logistic Regression | 0.8439 | 0.8578 | 0.4908 | 0.7252 | 0.5854 | 0.4936 |
| 6 | Linear Discriminant Analysis | 0.842 | 0.8559 | 0.501 | 0.71 | 0.5875 | 0.4934 |
| 7 | Ridge Classifier | 0.8419 | 0 | 0.4396 | 0.7532 | 0.5552 | 0.467 |
| 8 | SVM - Linear Kernel | 0.8407 | 0 | 0.4573 | 0.7331 | 0.5631 | 0.4721 |
| 9 | Ada Boost Classifier | 0.8383 | 0.8511 | 0.474 | 0.7091 | 0.5682 | 0.4735 |
| 10 | Random Forest Classifier | 0.8353 | 0.8309 | 0.4122 | 0.7385 | 0.529 | 0.4388 |
| 11 | K Neighbors Classifier | 0.8342 | 0.8178 | 0.4523 | 0.703 | 0.5505 | 0.4546 |
| 12 | Decision Tree Classifier | 0.7822 | 0.6882 | 0.5175 | 0.5148 | 0.5162 | 0.3756 |
| 13 | Naive Bayes | 0.6258 | 0.7112 | 0.6883 | 0.3369 | 0.4523 | 0.216 |
| 14 | Quadratic Discriminant Analysis | 0.5817 | 0.6343 | 0.6291 | 0.2959 | 0.4009 | 0.1396 |

| | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|
| 0 | 0.8417 | 0.8481 | 0.4384 | 0.7535 | 0.5543 | 0.4661 |
| 1 | 0.8418 | 0.8514 | 0.4380 | 0.7541 | 0.5542 | 0.4661 |
| Mean | 0.8418 | 0.8497 | 0.4382 | 0.7538 | 0.5542 | 0.4661 |
| SD | 0.0000 | 0.0016 | 0.0002 | 0.0003 | 0.0001 | 0.0000 |

| | Model | Accuracy | AUC | Recall | Prec. | F1 | Kappa |
|---|---|---|---|---|---|---|---|
| 0 | K Neighbors Classifier | 0.846 | 0.8605 | 0.4647 | 0.7553 | 0.5754 | 0.4878 |




We can see from this experiment that on dataset
2 the Tuned KNN performed better as compared to KNN. Both are very well calibrated and so we don't need to do any calibration. The Tuned KNN used Euclidean distance with K=10. We also KNN performance when compared to other algorithms and again see that CatBoost performs Best.

Comparison of Both Algorithms:
Dataset 1 - ANN Performed Best, Accuracy was the Measure and it is a balanced dataset.
Dataset 2 - ANN Performed Best,

KNN took lot of time to train as compared to ANN, which provided higher accuracy in lesser time. This is because KNN takes more time when features are more. So when we dummy our variables the cost of computation is higher for KNN.

We can improve performance in Dataset2 by using synthetic sampling like SMOTE, ADASYN as it is imbalanced.