

Empirical Learning algorithms Comparison

Yiou Lyu
Yiyi Xu
Shuyang Zhang

yil093@ucsd.edu
y3xu@ucsd.edu
s4zhang@ucsd.edu

Abstract

This paper is a replication of An Empirical Comparison of Supervised Learning Algorithms by Caruana and Niculescu-Mizil (Caruana, 2006). We aim to compare the algorithms' performances on binary classification between six supervised learning methods: logistic regression, SVM, kNN, naive bayes, random forest, and decision trees. Using grid search on hyperparameter tuning, training classifiers are optimized by three metrics, including accuracy, f1 and area under curve. Overall performances of the algorithms are compared through the average of the three metrics across five trials on each of the seven dataset. The support vector machine provides the best training accuracy across datasets, and random forest provides the best accuracy on the testing performance across different datasets. We compare the accuracy scores among the different algorithms by implementing the t-test. By averaging the solution of p values by dataset, we found that difference of performance between log vs knn, log vs nb, log vs dt, svm vs knn, svm vs nb, svm vs dt, and knn vs dt are statistically significant because their p-values are under the threshold of 0.05. It means in each of those pairs of algorithms, there is a significant difference between the performance accuracy of one algorithm against the other.

Keywords: Binary Classification, Logistic Regression, Support Vector Machines, k-Nearest Neighbors, Naive Bayes, Random Forest, Decision Trees

1. Introduction

STATLOG (King et al., 1995) study is one of the typical examples in the field of empirical studies comparing learning algorithms. Another predecessor paper Caruana and Niculescu-Mizil (2006), henceforth referred to as CNM2006, continues the main methodology of STATLOG and enlarged the algorithm pool with several newly developed learning algorithms. Those freshly added algorithms in CNM2006, like SVMs and random forests, exhibit different performance in various datasets and algorithm settings. Thus, empirical experimentation of algorithms allows the comparison among algorithms with data-oriented analysis.

Another breakthrough of CNM2006 is its thorough performance metrics. The paper creatively includes threshold metrics like F-scores and accuracy in order to measure if the performance is above the threshold of true prediction. Other evaluation metrics are ordering/ranking metrics and

probability metrics. For example, ROC of ordering metrics reflects true positive and false negative rate at different classification thresholds. A total of eight algorithm categories --SVMs, ANN, Logistic Regression, Naive Baye, KNN, Random Forest, Decision Trees, Bagged trees --are discussed there, and the comparison of them rely on the data from the performance metrics and calibration methods. To sum up, CNM2006 grants an adequate background knowledge of diving in supervised learning algorithms in the aspect of empirical assessment.

Following the lead of the elders, the study discussed in this report picks six algorithms of interest on binary classification, and replicates CNM06 methodology on seven data sets of real world with three performance metrics in specific. The reduced number of algorithms and performance metrics narrows the scope of evaluation data, therefore it further ensures the validity of pattern checking to the CNM2006.

2. Methodology

2.1 Learning Algorithms

All six algorithms are tuned using 5-fold cross validation by accuracy, F1 score and AUC. Parameters used in grid search or pipeline are detailed below for each learning algorithm.

Logistic Regression: Classifiers are trained using L1 regularized, L2 regularized and unregularized models. Regularization parameters are factors of 10 from 10^{-4} to 10^4 , combined with “saga” and “lbfgs” solvers depending on applicable regularizations. Max iterations are set to be 5000.

SVM: We use “rbf” kernels of support vector classification provided in scikit learn. Kernel coefficients used are 1e-6, 1e-5, 1e-4, 1e-3, and 1e-2. Regularization parameters are multiple of 10 from 1 to 10000.

KNN: We include 10 values of K ranging from K = 1 to K = 10 with uniform weights(All points in each neighborhood are weighted equally.), as well as 10 values of K ranging from K = 1 to K = 10 with distance weights and with p value from p=1 to p=5(where p represents Power parameter for the Minkowski metric, When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.).

Naive Bayes: We use gaussian naive bayes, and parameter var_smoothing is set to be iterating through 1e-9,2e-9,2e-9 to choose the best one during grid search of hyperparameter tuning. Var_smoothing is the portion of the largest variance of all features that is added to variances for calculation stability.

Random Forest: Includes two criterions - entropy and gini - for the random forest classifier. Three max features, auto, square root and log2 are applied here. The max depth has a range from

4 to 8 integrally. It is the maximum depth of the tree. Also, two numbers 200 and 500 are adopted here as the n_estimators, which is the number of trees in the forest. All of the variables mentioned above involve the hyperparameter tuning through GridSearchCV method.

Decision Trees: Similar to the random forest algorithm, the method adopted entropy or gini as criterions of the tree classifier. Then, the max depth the tree can reach is set in the range of a list of numbers: 40,45, 60 and 70. PCA and Standard Scaler are used as parameters of the pipeline to tune these two hyperparameters.

2.2 Performance Metrics

Model performance is measured using 3 performance metrics in the range of [0,1], which are accuracy, F1 score and Area Under Curve(AUC). They are used in both the grid search and the testing predictions.

Accuracy calculates the ratio between true positives added by true negatives, and the overall number of samples,

F1 score is calculated by $F1 = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$. As a widely used metric in machine learning, it works well in the binary classification. (Chicco, Jurman, 2020).

Area Under the ROC curve metric measures the ordering of cases, regardless of the actual predicted values (Caruana, 2006).

2.3 Comparing Across Performance Metrics

The three performance metrics have range [0,1]. Higher values are better. While accuracy has baseline rates that depend on the data, area under the ROC curve have their own baseline rate independent of the data (Caruana, 2006). Since we are comparing the analysis across algorithms in this paper, we disregard the baseline rate. Performances would be assessed by comparing the three performance metrics across algorithms.

2.4 Data Sets

Seven Data sets are selected from UCI Machine Learning Repository (Blake, Merz, 1998). They are EGSS, AI4I, ODDS, IFDD, EEG, DefaultC, HTRU (Table 1). For categorical features in the datasets, we applied one-hot encoding. None of the datasets contain missing values.

For the EGSS dataset, binary labels are stable (1) and unstable (0). It is a relatively balanced dataset.

For the AI4I dataset, binary labels are machine failure (1), and not failure (0). This dataset is highly imbalanced with a label 1 proportion of 3.39%.

For the ODDS dataset, binary labels are occupied (1) and not occupied (0). It is an imbalanced dataset,

For the IFDD dataset, binary labels are action predictions of allow (1) and not allow (0). It is a balanced dataset. The original dataset has 4 actions: “allow”, “drop”, “deny”, and “reset-both”. Since we are testing the algorithms’ performances on binary classification, we categorized the actions into allow actions and actions other than allowing. In the original dataset, the first four columns represent ports id, which does not count as a feature. Thus, we deleted the first four columns in our classification.

For the EEG dataset, binary labels are eye-closed state (1) and eye-open state (0). It is a balanced dataset.

For the DefaultC dataset, binary labels are response variables yes (1) and no (0). It is an unbalanced dataset.

For the HTRU dataset, binary labels are classes of pulsar candidates 0 and 1. It is a highly unbalanced dataset.

Table1: Datasets explanation

No.	acronym	dataset name	# attributes	training size	test size	% proportion of label 1 with respect to whole
1	EGSS	Electrical Grid and Stability Stimulated Dataset	14	6700	3300	36.20%
2	AI4I	AI4I 2020 Predictive Maintenance Dataset	14	6700	3300	3.39%
3	ODDS	Occupancy Detection Dataset	7	13775	6785	13.14%
4	IFDD	Internet Firewall Data Dataset	12	43906	21626	57.44%
5	EEG	EEG Eye State Dataset	15	10037	4943	44.88%
6	DefaultC	Default of credit card clients Dataset	24	20100	9900	22.12%
7	HTRU	HTRU2 Dataset	9	11992	5906	9.16%

3. Experiment

Our first step was to take a look at the datasets, and then did data cleaning on picking the reasonable columns for prediction. For each dataset, we randomly extracted 0.33 of the data as

the testing set and the rest as training set with an exception for the IFDD and DefaultC datasets used in SVM. Training data of 5000 are applied to both of the datasets for SVM due to time constraints. The training and testing dataset split is applied 5 times for each trial, with an assigned random seed from 1 to 5. The assigned random seed allows for replication of the experiment.

Six algorithms are trained with the splitted dataset. For each of the train/test splits, a 5-fold cross validation in gridsearch is applied for the hyperparameter search. Parameter spaces are described above in the 2.1 Learning Algorithm section. For the grid search, we applied three scoring including accuracy, f1 score and AUC. Hyperparameter settings for the three highest scoring are used to train the classifier, and applied to the testing set to obtain a score of the performance metrics.

After the main experiment is finished, we tried feature selection because we wish to improve the time efficiency for algorithms. Feature selection is performed for each trial after training and testing data split. We used L1 based feature selection using SelectFromModel from sklearn.feature_selection. However, since we successfully implemented this process rather late, due to time constraints, we are unable to use the feature selected into the classifier training process.

4. Performance

4.1 Performances by Metric

Main results of this experiment are constructed as performance by metrics shown in table 2. We average the performance metrics across trials and across dataset to obtain a table showing each algorithm's performance in three metrics. Algorithm with the best performance on each metrics is **boldfaced**. Other algorithms whose performance are not statistically distinguishable from the best using threshold $p=0.05$ are marked with a *. The p values are calculated using paired t-test on 5 trials. Mean normalized score is also calculated over three metrics for comparison of 6 algorithms overall.

For Accuracy, SVM scored the highest of 0.95213. For both F1-score and ROC_AUC score, Random Forest scored the highest as 0.93571 and 0.87737. Random Forest also scored the highest in mean performance across metrics. All algorithms have performances that are not statistically distinguishable from the best.

Table 2: test set Performance over metrics(average over 7 datasets)

Model	Accuracy score	F1 - Score	ROC_AUC Score	Mean Performance Across Metrics
Logistic Regression	0.88600*	0.67963*	0.78562*	0.78375*

Support Vector Machine	0.95213	0.78422*	0.87390*	0.87009*
K-Nearest Neighbors	0.81818*	0.78028*	0.73610*	0.77819*
Naive Bayes	0.68203*	0.622*	0.70388*	0.66930*
Decision Tree	0.90426*	0.88562*	0.82738*	0.87242*
Random Forest	0.92849*	0.93571	0.87737	0.91386

4.2 Performances by Problem

Test set performance by datasets are shown in Table 3. We average the performance metrics across trials and across performance metrics to obtain a table showing each algorithm's performance in the 7 datasets. Algorithm with the best performance on each dataset is **boldfaced**. Other algorithms whose performance are not statistically distinguishable from the best using threshold p=0.05 are marked with a *. The p values are calculated using paired t-test on 5 trials.

Support Vector Machine scores the highest in dataset 3, 4, and 5 with 0.98209, 0.99726, and 0.97522 respectively. Naive Bayes scores the highest in dataset 2 with 0.99575. Random Forest scores the highest in dataset 1 and 7, with 0.99982 and 0.95109 respectively. All algorithms have performances that are not statistically distinguishable from the best.

Table 3: test set Performance by problem(average over three metrics)

	dataset1	dataset2	dataset3	dataset4	dataset5	dataset6	dataset7
Logistic Regression	0.78059*	0.61320*	0.98127*	0.99583*	0.60502*	0.58841*	0.92197*
Support Vector Machine	0.96455*	0.80298*	0.98209	0.99726	0.97522*	0.44957*	0.91895*
K-Nearest Neighbors	0.81043*	0.99562*	0.97147*	0.55943*	0.49117*	0.68603*	0.93319*
Naive Bayes	0.34927*	0.99575	0.97404*	0.56060*	0.43589*	0.44065*	0.92892*
Decision Tree	0.89914*	0.88853*	0.87565*	0.85735*	0.84739*	0.84201	0.89689*
Random Forest	0.99982	0.99066*	0.96909*	0.94251*	0.79091*	0.75293*	0.95109

5. Discussion

As shown in table 1, dataset 4 and 5 are the most balanced dataset. Dataset 2 and 7 are the most imbalanced dataset. Comparing the experiment result of performance by problem in tables, Support Vector Machine has the highest mean score for both 4 and 5, suggesting that Support Vector Machine has good performance for balanced dataset. Both Random Forest and Naïve Bayes score the highest in dataset 7 and 2 respectively, suggesting that they have good performances over imbalanced datasets. K-NN does not show apparent advantage over other algorithms overall. In general, random forest performs better than decision tree across all the datasets except for dataset 5. By taking a close look at Table 3, K-NN and Naïve Bayes do worse on balanced datasets than other algorithms. This could be due to the improper range of k or var_smoothing, or the limitation of the algorithms themselves.

For tree algorithms, on one hand , Decision Tree does not show significant differences across seven datasets, which all ranges from [0.84,0.90]. This suggests that features of datasets, including its balance, number of attributes, may not influence much on Decision Tree's performance. Its performance is the most stable one that as a whole in terms of numerical data range. On the other hand, another tree algorithm random forest scores the highest on average in Table 2, which suggests that Random Forest also scored the highest in mean performance across metrics. In Table 3, its performance beat Decision Tree six out of seven times. Thus, we are able to conclude that usually random Forest is better than decision tree in application.

Overall, the support vector machine provides the best training accuracy across datasets, and random forest provides the best accuracy on the TESTING performance across different datasets according to its average performance on the table3. For individual dataset, random forest does not necessarily perform the best among algorithms on TESTING, like its performance on dataset 2, 3, 4, 5, and 6. On dataset1, random forest provides the most accurate result on both the training and test data. On dataset 2, KNN provides the most accurate training accuracy but Naïve Bayes provides the most accurate testing accuracy. Overall they all perform well on training with accuracy between 0.967~0.998, but on testing they differ greatly, with logistic regression having the lowest performance score 0.612. On dataset3, all algorithms perform well on training with accuracy between 0.969~0.997, and on testing all algorithms perform well with high accuracy except for decision tree. On dataset 4, k nearest neighbors and naive bayes perform poorly with training accuracy 0.647 and 0.671, and they also perform poorly with testing accuracy scores with 0.559 and 0.560, while other algorithms all reach high accuracy scores. For dataset 5,The logistic regression performs poorly on the training and testing. For dataset 6, logistic regression has relatively high training accuracy but low testing accuracy, only the decision tree has relatively high testing accuracy. For dataset 6, logistic regression has relatively high training accuracy but low testing accuracy, only random forest has relatively high testing accuracy.

We compare the accuracy scores among the different algorithms by implementing the t-test and get the solution of t-test scores and the corresponding p-values, in order to see if algorithms' performance has statistically significant differences. We set the p threshold to 0.05. According to the appendix 3.1-3.4 tables, we found that there is a significant difference between their

performance of logistic regression and support vector machines in all datasets except for dataset 3, 4, 7. For logistic regression versus k-nearest neighbors, there is a significant difference between their performance in all datasets because the p value is way smaller than 0.05. For logistic regression versus naive bayes, there is a significant difference between their performance in all datasets because the p value is way smaller than 0.05. For logistic regression versus decision tree, there is a significant difference between their performance in all datasets because the p value is way smaller than 0.05. For logistic regression versus random forest, there is a significant difference between their performance in all datasets except for dataset 2, 3, 4, 6, 7. For support vector machines versus k-nearest neighbors, there is a significant difference between their performance in all datasets because the p value is way smaller than 0.05. For support vector machines versus naive bayes, there is a significant difference between their performance in all datasets. For support vector machines versus decision trees, there is a significant difference between their performance in all datasets except for dataset 4.

For support vector machines versus random forest, there is a significant difference between their performance in all datasets except for dataset 2, 3, 4, 7 . For kNN versus naive bayes, there is a significant difference between their performance in all datasets except for dataset 2, 3. For kNN versus decision trees, there is a significant difference between their performance in all datasets. For kNN versus random forest, there is a significant difference between their performance in all datasets except for dataset 2, 3, 7. For naive bayes versus decision tree, there is a significant difference between their performance in all datasets except for dataset 2. For naive bayes versus random forest, there is a significant difference between their performance in all datasets except for dataset 2, 3. For decision tree versus random forests, there is a significant difference between their performance in all datasets except for dataset 2, 3, 4, 5.

By averaging the p values by dataset, we found that difference of performance between log vs knn, log vs nb, log vs dt, svm vs knn, svm vs nb, svm vs dt, and knn vs dt are statistically significant because their p-values are under the threshold of 0.05. It means in each of those pairs of algorithms, there is a significant difference between the performance accuracy of one algorithm against the other.

We included a feature selection part at the end of the code, and found that for some of the datasets, multiple features can be eliminated from the training to make the fitting process more efficient. 9 features are selected from 12 features in dataset 1 across all 5 trials. 5 features are selected from 6 features in dataset 2 across all 5 trials. 4 features are selected from 5 features in dataset 3 across all 5 trials. All features are selected for dataset 4 for all trials. 13, 12, or 11 features out of 14 features are selected in different trials for dataset 5. 9 or 10 features are selected from 23 features in dataset 6 for different trials. 7 or 8 features are selected from 8 features in dataset 10 for different trials. This also suggests that different features might be selected for the same dataset with different splits of training and testing sets. By eliminating some features from the dataset, feature selection reduces the complexity of some dataset. However, due to time constraints, we are unable to use the feature selection into our training

process, In the future study, if future readers are interested in performing similar research, we recommend them to learn or replicate our feature selection in order to more efficiently implement their algorithms and projects.

6. Bonus Points

In addition to the required amount of algorithms*datasets (3 people * 3 algorithms * 4 datasets * 5 trials = 180), we performed in total of 210 (6 algorithms * 7 datasets * 5 trials = 210) in order to get a more comprehensive analysis of the datasets. Other than the logistic regression, svm, kNN, random forest and decision trees taught in class, we self-explored the algorithm of naive bayes as it is a very useful and widely used algorithm in supervised learning.

We have put tremendous effort in collecting the dataset for use because for our first draft of the project proposal, the datasets include data that only contains text instead of numbers, and some datasets do not have enough instances. Therefore we decided that it will be a trouble to convert each feature into the form of algebraic numbers. After three times turning down the whole proposal for collecting datasets, we finally decided the proper datasets for use. We picked the dataset that works the most intuitively.

We included feature selection which is appended at the end of the code because at first we did not quite understand how feature selection works, but then after we finished the project and tried, we found that feature selection actually works and would reduce a lot of our workload if implemented earlier before. Even though we did not have the time to restart over again, we are really interested in the convenience of implementing the feature selection part. So we included feature selection results separately from our original project in the discussion and the code, in order to make some suggestions to our readers and inform our readers that, if you are interested in replicating our paper or doing your own project, you can implement the feature selection like we did.

References

- Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.
- Caruana, R. and Niculescu-Mizil, A., 2006. An Empirical Comparison Of Supervised Learning Algorithms. In Proceedings of the 23rd international conference on Machine Learning, 161-168.
- Chicco, D., Jurman, G. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics* 21, 6 (2020).
<https://doi.org/10.1186/s12864-019-6413-7>

Appendix

Appendix Table 1.1 Accuracy Score of different algorithms on both datasets and trials

Model		Accuracy score						
		dataset1	dataset2	dataset3	dataset4	dataset5	dataset6	dataset7
Logistic Regression	trial1	0.813636363 6	0.97090909 09	0.98570227 08	0.9984278 184	0.63450647 25	0.8072727 273	0.98036228 2
	trial2	0.823636363 6	0.97121212 12	0.98373983 74	0.9989364 654	0.64057443 37	0.8122222 222	0.97866937 53
	trial3	0.826969697	0.97	0.98345948 98	0.9987977 435	0.63430420 71	0.8180808 081	0.97782292 2
	trial4		0.96939393 94	0.98654331 37	0.9990751 873	0.64340614 89	0.8052525 253	0.98019299 14
	trial5	0.809393939 4	0.96454545 45	0.98850574 71	0.9986127 809	0.65068770 23	0.8084848 485	0.97968511 93
Support Vector Machine	trial1	0.966969697	0.97696969 7	0.98654331 37	0.9972576 488	0.97977346 28	0.77696	0.97782292 2
	trial2	0.971212121 2	0.97909090 91	0.98486122 79	0.9979514 967	0.97431229 77	0.77928	0.98002370 07
	trial3	0.970303030 3	0.97939393 94	0.98430053 27	0.9988931 474	0.97491909 39	0.77768	0.97883866 6
	trial4	0.968787878 8	0.97424242 42		0.9984966 629	0.97653721 68	0.77992	0.97968511 93
	trial5	0.970909090 9	0.97696969 7	0.98542192 32	0.9987609 859	0.97552589	0.7764	0.98036228 2
K-Nearest Neighbors	trial1	0.837575757 6	0.99939393 94	0.96663863 19	0.6387878 788	0.55521844 66	0.7869696 97	0.97325207 38
	trial2	0.835757575 8	0.99969696 97	0.97252593 22	0.6430303 03	0.47997572 82	0.7980808 081	0.96969696 97
	trial3	0.842424242 4	0.99909090 91	0.95934959 35	0.6345454 545	0.47451456 31	0.7966666 667	0.97138987 64
	trial4	0.832121212 1	0.99848484 85	0.98121670 87	0.6378787 879	0.52568770 23	0.7835353 535	0.97392923 65
	trial5	0.835757575 8	0.99909090 91	0.98317914 21	0.6293939 394	0.55987055 02	0.79111111 11	0.97071271 37
Naive Bayes	trial1	0.359696969 7	0.99969696 97	0.97140454 16	0.6578787 879	0.49474110 03	0.4129292 929	0.94633485 69

	trial2	0.359696969 7	0.99757575 76	0.97028315 11	0.6681818 182	0.44822006 47	0.3814141 414	0.94430336 89
	trial3	0.354848484 8	0.99757575 76	0.97252593 22	0.6475757 576	0.46237864 08	0.3608080 808	0.94430336 89
	trial4	0.357575757 6	0.99878787 88	0.97196523 69	0.6630303 03	0.45307443 37	0.3782828 283	0.94735060 1
	trial5	0.357272727 3	0.99757575 76	0.97308662 74	0.6524242 424	0.44599514 56	0.3771717 172	0.94531911 29
Decision Tree	trial1	0.999696969 7	0.99818181 82	0.99712184 2	0.9977342 088	0.83737864 08	0.7363636 364	0.96207888 95
	trial2	0.999696969 7	0.99666666 67	0.99392388 87	0.9980116 526	0.83394012 94	0.7327272 727	0.96089385 47
	trial3	0.999696969 7	0.99848484 85	0.99616245 6	0.9977342 088	0.83798543 69	0.7322222 222	0.96343321 48
	trial4	0.999696969 7	0.99787878 79	0.99712184 2	0.9984278 184	0.82160194 17	0.7293939 394	0.96106314 54
	trial5	0.999696969 7	0.99818181 82	0.99360409 34	0.9977342 088	0.82301779 94	0.7566666 667	0.96427966 82
Random Forest	trial1	0.999696969 7	0.99818181 82	0.99488327 47	0.9975030 056	0.72673948 22	0.8206060 606	0.97917724 73
	trial2	0.999696969 7	0.99666666 67	0.99216501 44	0.9987977 435	0.83353559 87	0.8178787 879	0.97782292 2
	trial3	0.999696969 7	0.99909090 91	0.99248480 97	0.9984278 184	0.75525889 97	0.8174747 475	0.97663788 73
	trial4	0.999696969 7	0.99969696 97	0.99424368 4	0.9989364 654	0.82746763 75	0.8284848 485	0.97562214 32
	trial5	0.999696969 7	0.93787878 79	0.64614646 63	0.9985665 403	0.83758090 61	0.8164646 465	0.96427966 82

Appendix Table 1.2 F1 Score of different algorithms on both datasets and trials

Model		F-1 score						
		dataset1	dataset2	dataset3	dataset4	dataset5	dataset6	dataset7
Logistic Regression	trial1	0.7313237 221	0.3424657 534	0.97199341 02	0.9986229 243	0.5444920 595	0.3420689 655	0.8851485 149
	trial2	0.7356948 229	0.3065693 431	0.96741573 03	0.9990820 196	0.5463364 82	0.3600688 468	0.8781431 335
	trial3	0.7285006 754	0.2877697 842	0.96863370 55	0.9989588 339	0.5443548 387	0.3606673 766	0.8704253 215

	trial4	0.7458834 001	0.2170542 636	0.97315436 24	0.9991978 181	0.5482961 824	0.3428766 19	0.8842729 97
	trial5	0.7210643 016	0.2251655 629	0.97782585 18	0.9987899 322	0.5570659 143	0.3515731 874	0.8762886 598
Support Vector Machine	trial1	0.9537547 73	0.6380952 381	0.97336293 01	0.9980194 041	0.9775078 722	0.0770857 0499	0.8678102 926
	trial2	0.9601342 845	0.6790697 674	0.96942242 36	0.9982163 406	0.9715692 859	0.0333971 1254	0.8856589 147
	trial3	0.9584393 554	0.6494845 361	0.96972972 97	0.9990372 318	0.9719710 669	0.0665897 4782	0.8766041 461
	trial4	0.9560017 087	0.6187845 304	0.97843359 82	0.9986931 484	0.9734675 206	0.0617283 9506	0.8726919 339
	trial5	0.9590443 686	0.6576576 577	0.97136563 88	0.9989199 775	0.9727784 027	0.0725557 1824	0.8809034 908
K-Nearest Neighbors	trial1	0.82	1	0.97	0.55	0.46	0.71	0.97
	trial2	0.82	1	0.97	0.56	0.37	0.74	0.97
	trial3	0.83	1	0.96	0.53	0.38	0.73	0.97
	trial4	0.82	1	0.98	0.54	0.51	0.71	0.97
	trial5	0.82	1	0.98	0.53	0.44	0.73	0.97
Naive Bayes	trial1	0.19	1	0.97	0.52	0.46	0.42	0.95
	trial2	0.19	1	0.97	0.54	0.3	0.38	0.95
	trial3	0.19	1	0.97	0.51	0.34	0.35	0.95
	trial4	0.19	1	0.97	0.53	0.32	0.37	0.95
	trial5	0.19	1	0.97	0.52	0.29	0.37	0.95
Decision Tree	trial1	1	1	1	1	0.84	0.74	0.96
	trial2	1	1	0.99	1	0.83	0.73	0.96
	trial3	1	1	1	1	0.84	0.73	0.96
	trial4	1	1	1	1	0.82	0.73	0.96
	trial5	1	1	0.99	1	0.82	0.75	0.96
Random Forest	trial1	1	1	0.99	1	0.72	0.79	0.98
	trial2	1	1	0.99	1	0.83	0.79	0.98

	trial3	1	1	0.99	1	0.75	0.79	0.98
	trial4	1	1	0.99	1	0.82	0.81	0.97
	trial5	1	1	0.99	1	0.84	0.79	0.96

Appendix Table 1.3 ROC_AUC Score of different algorithms on both datasets and trials

Model	ROC_AUC score							
		dataset1	dataset2	dataset3	dataset4	dataset5	dataset6	dataset7
Logistic Regression	trial1	0.789862 4514	0.6085505 666	0.9872877 867	0.9713476 325	0.6207840 543	0.5999258 601	0.9165333 609
		0.792687 0646	0.6164493 891	0.9850099 762	0.9942157 332	0.6260093 249	0.6069904 094	0.9072324 809
	trial3	0.791735 1735	0.5908497 704	0.9856165 886	0.9946369 967	0.6203768 107	0.6074565 949	0.9009310 987
		0.800747 5216	0.5735931 759	0.9863051 267	0.9942270 686	0.6275254 345	0.5995760 034	0.90784117 02
	trial5	0.780475 8666	0.58351128 37	0.9878771 364	0.9944771 426	0.6365178 07	0.6035610 274	0.9060650 729
Support Vector Machine	trial1	0.962814 1433	0.7893084 878	0.9884126 461	0.9955914 399	0.9797267 138	0.51107259 18	0.8994540 456
		0.969582 3304	0.8131526 858	0.9869330 103	0.9947032 114	0.9739061 642	0.5038629 491	0.9039220 957
	trial3	0.969107 0288	0.7736478 673	0.9876530 1	0.9947953 041	0.9746795 411	0.5094199 691	0.8918994 413
		0.964246 882	0.76117661 65	0.9881604 143	0.9947564 034	0.9760848 459	0.5080069 346	0.9122260 717
	trial5	0.967009 8962	0.7777723 853	0.9874812 657	0.9948108 842	0.9755358 258	0.5095367 435	0.8963029 597
K-Nearest Neighbors	trial1	0.774959 1229	0.9912280 702	0.9582443 205	0.5029205 743	0.5124980 672	0.5355874 409	0.8619229 128
		0.772247 1434	0.9956521 739	0.9767506 788	0.5037057 745	0.5200760 3	0.5618814 235	0.8500716 872
	trial3	0.779312 4832	0.9868421 053	0.9443446 212	0.5024799 514	0.5174849 235	0.5356048 056	0.8543761 639
		0.765817 8766	0.9764150 943	0.9833030 502	0.4936368 09	0.5492105 802	0.5284901 699	0.8680996 896
	trial5	0.770520	0.9884615	0.9865424	0.4951824	0.5130059	0.5525363	0.8542275

		8907	385	749	08	18	889	491
Naive Bayes	trial1	0.5	0.9956140 351	0.9798287 677	0.5	0.52114953 84	0.5745453 048	0.89614211 57
	trial2	0.5	0.9945532 728	0.9796104 822	0.5	0.4931529 379	0.5631989 28	0.8882281 508
	trial3	0.5	0.9860574 223	0.9808131 829	0.5	0.5094450 907	0.5536558 474	0.8830540 037
	trial4	0.5	0.98113207 55	0.9806713 928	0.5	0.5049855 779	0.5585645 241	0.8971508 127
	trial5	0.5	0.9876728 949	0.9805138 592	0.5	0.4952736 43	0.5591384 576	0.8916726 745
Decision Tree	trial1	0.999761 7913	0.9714285 714	0.9958699 814	0.8349207 526	0.8359207 526	0.6196875 127	0.8944705 367
	trial2	0.999763 3696	0.9942016 057	0.9906682 158	0.8346592 729	0.8316592 729	0.6198171 216	0.8897778 268
	trial3	0.999763 3696	0.9951812 163	0.9945064 557	0.8329989 419	0.8359989 419	0.6142853 915	0.8946233 536
	trial4	0.999763 3696	0.9865282 321	0.9953891 089	0.8183748 352	0.8193748 352	0.6106893 58	0.8880819 367
	trial5	0.999763 3696	0.98113207 55	0.9901278 785	0.8229109 904	0.8219109 904	0.6317335 888	0.8989023 765
Random Forest	trial1	0.999761 7913	0.9714285 714	0.9954500 042	0.8329207 526	0.7122587 678	0.6356943 047	0.9085420 695
	trial2	0.999763 3696	0.9940446 691	0.9915576 032	0.8295927 292	0.8230078 844	0.6477040 171	0.9058422 493
	trial3	0.999763 3696	0.9953382 022	0.9943862 393	0.8329989 419	0.7444681 564	0.6331857 689	0.9101700 1
	trial4	0.999763 3696	0.9863712 955	0.9943976 276	0.8287483 523	0.8161713 635	0.6585245 915	0.8994413 408
	trial5	0.999763 3696	0.98113207 55	0.9907277 301	0.82110990 36	0.8271883 257	0.6478991 022	0.8989023 765

Appendix Table 2 Mean training set performance for the optimal hyperparameters on each dataset/algorithm combo with discussion

Appendix Table 3.1 t- test on accuracy score on TESTING over 5 trials on 7 datasets: t values

Appendix Table 10: Test on accuracy score on TIMIT dataset with different models.

		vs svm	vs knn	vs nb	vs dt	vs rf	vs knn	vs nb	vs dt	vs rf	n vs nb	vs dt	n vs rf	vs dt	vs rf	vs rf
data set1	EGSS	-46.1	-4.40	142.8	-56.6	-56.6	74.71	445.9	-38.2	-38.2	210.5	-97.1	-97.1	-712.	-712.	
		7890	4815	2586	0017	0017	55491	2250	1041	1041	1426	57519	5751	4277	4277	
		62	958	52	979	979		24	268	268	38	44	944	502	502	
data set2	AI4I	-6.12	-25.8	-25.7	-21.0	-1.55	-27.8	-17.0	-19.8	-0.74	1.792	2.806	1.060	0.768	0.99	0.9
		5814	2179	5119	6909	7939	16026	9052	7253	2262	8429	24304	8755	2212	8906	498
		243	348	941	452	723	47	043	187	7115	14		83	796	6888	961
data set3	ODDS	-0.52	3.469	17.89	-7.69	0.87	3.331	14.06	-12.4	0.896	0.165	-4.80	0.673	-26.2	0.68	1.0
		0919	4558	2270	9344	7709	35510	7758	3960	3264	8846	81967	1683	7240	6087	382
		1815	64	19	427	3367	4	98	168	126	205	78	886	856	0726	451
data set4	IFDD	1.839	161.4	94.35	10.9	2.02	145.6	88.97	1.075	-0.75	-9.17	-162.	-157.	-95.0	-94.7	-2.6
		9821	8638	9618	9502	9198	52247	7947	0499	2921	3791	25527	7785	7017	4990	516
		29	49	63	126	429	5	02	49	653	172	93	219	305	616	768
data set5	EEG	-96.0	7.185	15.91	-29.7	-7.58	26.22	63.69	41.34	7.620	3.321	-15.6	-9.33	-50.3	-10.6	1.3
		6913	1492	5398	6466	3442	48297	0223	6428	4896	6806	39778	5612	6774	8999	432
		974	9	02	023	907	3	47	16	24	07	61	404	398	808	725
data set6	Default C	13.23	13.23	42.71	12.5	-2.47	-4.54	45.66	7.451	-23.2	40.50	9.572	-6.15	-36.5	-51.9	-13.
		2289	7467	3024	8978	3590	70445	7073	3857	1251	5473	17223	6988	1708	7280	011
		77	19	16	558	88	5	17	46	792	45	2	936	56	859	684
data set7	HTRU	0	12.63	116.5	18.9	1.67	6.860	41.53	21.82	1.550	71.28	8.223	-1.13	-16.5	-10.7	-3.8
			5017	2573	1983	0731	72661	4961	6086	0772	5720	6864	9747	0716	4858	920
			49	03	695	706	7	47	86	74	53		877	471	088	189

Appendix Table 3.2 t- test on accuracy score on TESTING over 5 trials on 7 datasets : P values

		p:log vs svm	p:log vs knn	p:log vs nb	p:log vs dt	p:log vs rf	p:svm vs knn	p:svm vs nb	p:svm vs dt	p:svm vs rf	p:knn vs nb	p:knn vs dt	p:knn vs rf	p:nb vs dt	p:nb vs rf	p:dt vs rf
data set1	EGSS	1.32E-06	0.011 64746 236	1.44E-08	5.83E-07	5.83E-07	1.92E-07	1.52E-10	2.80E-06	2.80E-06	3.05E-09	6.73E-08	6.73E-08	2.33E-11	2.33E-11	
data set2	AI4I	0.0035 977153 09	1.34E-05	1.35E-05	3.00E-05	0.194 24609 87	9.94E-06	6.88E-05	3.78E-05	0.4991 490174	0.147461 6845	0.0485 02074 88	0.3485 54204 88	0.485 19796 33	0.374 37060 88	0.39 5954 0982
data set3	ODDS	0.6299 306787	0.025 59629 342	5.73E-05	0.001 53111	0.429 65878	0.0290 689087	0.0001 481704	0.0002 401280	0.4207 460622	0.876294 6647	0.0085 96602 92	0.5377 317785	1.25E-05 16	0.530 35901 694	0.35 7793
data	IFDD	0.1396	8.82E-00	7.56E-00	0.000	0.112	1.33E-00	9.56E-00	0.3428	0.4933	0.000783	8.65E-00	9.68E-00	7.34E-00	7.44E-00	0.05

set4		013314	-09	08	38885 60871	31511 49	08	08	762216	843235	9943013	09	09	-08	08	6882 1222 7
data set5	EEG	7.04E-08	0.001 98750 6692	9.11E-05	7.59E-06	0.001 62160 6369	1.26E-05	3.64E-07	2.05E-06	0.0015 919799 93	0.029332 46126	9.76E-05 329405 898	0.0007 329405	9.30E-07 43383 06329	0.000 0.25 0332 4207	
data set6	Defaut C	0.0001 884750 515	0.000 18818 58436	1.80E-06 22910 18932	0.000 68156 419	0.0104 410288 6	1.38E-06 329133 65	0.0017 65507	2.04E-05 311501 7	2.22E-06 628	0.0006 311501	0.0035 7	3.36E-06 8.20E-07 0.00	8.20E-07 0.00 0.00	0.00 0.201 3291 265	
data set7	HTRU	1	0.000 22590 51349	3.25E-08 4.60E-05	0.170 09501 26	0.0023 633797 28	2.01E-06 565573	2.61E-05 0.1960	2.32E-07 565573	0.0011 91905 919	0.3180 073782	7.89E-05 7.89 0.01	0.000 42471 02536	0.000 7660 3779 2		
averages cross datasets			0.00 5665												0.1 798	
		2.53E-01	5320 59	2.34 E-05	3.19 E-04	1.40 E-01	5.99E-03 -03	3.15E-05 -05	4.93E-02 -02	2.30E-01 -01	1.51E-01 8.44 E-03	1.73E-01 6.93 E-02	1.29 E-01 0.07	0.000 42471 02536	0.000 7660 3779 2	

Appendix Table 3.3 t- test on accuracy score on TRAINING over 5 trials on 7 datasets: t values

		t:log vs svm	t:log vs knn	t:log vs nb	t:log vs dt	t:log vs rf	t:svm vs knn	t:svm vs nb	t:svm vs dt	t:svm vs rf	t:knn vs nb	t:knn vs dt	t:knn vs rf	t:nb vs dt	t:nb vs rf	t:dt vs rf
data set1	EGSS	-46.1 7890 62	-4.40 4815 958	142. 8258 652	-56.6 0017 979	-56.6 0017 979	74.71 55491	445.92 25024	-38.21 04126 8	-38.21 04126 8	210.51 42638	-97.15 75194 4	-97.15 751944	-712.4 27750 2	-712. 42775 02	
data set2	AI4I	-6.12 5814 243	-25.8 2179 348	-25. 7511 9941	-21.0 6909 452	-1.55 7939 723	-27.81 60264 7	-17.09 05204 3	-19.87 25318 7	-0.742 26271 15	1.7928 42914	2.8062 4304	1.0608 75583	0.768 22127 96	0.998 90668 88	0.949896 1467
data set3	ODDS	-0.52 0919 1815	3.46 9455 864	17.8 9227 019	-7.69 9344 427	0.87 7709 3367	3.331 35510 4	14.067 75898	-12.43 96016 8	0.8963 26412 6	0.1658 84620 5	-4.808 19677 8	0.6731 683886 8	-26.27 24085 6	0.686 08707 26	1.038245 132
data set4	IFDD	1.83 9982 129	161. 4863 849	94.3 5961 863	10.99 5021 26	2.02 9198 429	145.6 52247 5	88.977 94702	1.075 04994 9	-0.752 92165 3	-9.173 79117 2	-162.2 55279 3	-157.7 785219 3	-95.07 01730 5	-94.7 49906 16	-2.65167 6814
data set5	EEG	-96.0 6913 974	7.18 5149 29	15.9 1539 802	-29.7 6466 023	-7.58 3442 907	26.22 48297 3	63.690 22347	41.34 64281 6	7.6204 89624 864	3.3216 80607 1	-15.63 97786 612404	-9.335 77439 8	-50.36 89998 08	-10.6 1.343272 564	
data set6	Defaut tC	13.2 3228 977	13.2 3746 719	42.7 1302 416	12.58 9785 58	-2.47 3590 88	-4.547 04455	45.667 07317	7.451 38574 6	-23.21 25179 2	40.505 47345	9.5721 72232 988936	-6.156 70856 72808	-36.51 59	-51.9 8471	-13.0116
data set7	HTRU	0	12.6 3501 749	116. 5257 303	18.91 9836 95	1.67 0731 706	6.860 72661 7	41.534 96147	21.82 60868 6	1.5500 77274 72053	71.285 864	8.2236 747877 864	-1.139 747877 1	-16.50 71647 88	-10.7 48580 8821	-3.89201

Appendix Table 3.4 t- test on accuracy score on TRAINING over 5 trials on 7 datasets: P values

		p:log vs svm	p:log vs knn	p:lo g vs nb	p:log vs dt	p:log vs rf	p:svm vs knn	p:svm vs nb	p:svm vs dt	p:svm vs rf	p:knn vs nb	p:knn vs dt	p:knn vs rf	p:nb vs dt	p:nb vs rf	p:dt vs rf
data set1	EGSS	1.32 E-06 47462 36	0.0116 47462	1.44 E-08	5.83E -07	5.83 E-07	1.92E- 07	1.52E- 10	2.80E- 06	2.80E- 06	3.05E- 09	6.73E- 08	6.73E- 08	2.33E- 11	2.33E- 11	
data set2	AI4I	0.003 5977 1530 9	1.34E- 05 5977	1.35 E-05	3.00E- 05	0.194 2460	9.94E- 06 987	6.88E- 05	3.78E- 05	0.499 14901	0.1474 61684	0.0485 02074	0.348 55420	0.485 19796	0.374 37060	0.3959 54098
data set3	ODD S	0.629 9306 787	0.0255 96293 42	5.73 E-05 53111	0.001 6587	0.429 68908	0.0290 48170	0.0001 40128	0.0002 74606	0.420 94664	0.8762 96602	0.0085 73177	0.537 73177	1.25E- 05 16	0.530 35901	0.3577 93694
data set4	IFDD	0.139 6013 314	8.82E- 09 38885	7.56 E-08 3151	0.000 60871	0.112 149	1.33E- 08 3151	9.56E- 08 149	0.3428 76221	0.493 38432	0.0007 83994	8.65E- 09 3013	9.68E- 09 3013	7.34E- 08 -08	7.44E- 08 -08	0.0568 82122 27
data set5	EEG	7.04 E-08 87506 692	0.0019 87506 692	9.11 E-05 -06	7.59E- 06 6216	0.001 0636	1.26E- 05 9	3.64E- 07	2.05E- 06	0.001 59197 9993	0.0293 32461 26	9.76E- 05 73294 05898	0.000 000 9.30E- 07 43383 06329	0.000 0.000 0.000 -07 32420 7	0.2503 06329 7	
data set6	Defau tC	0.000 1884 7505 15	0.0001 88185 8436	1.80 E-06 18932	0.000 22910 6419	0.068 6815 86	0.0104 41028	1.38E- 06 365	0.0017 32913	2.04E- 05 73	2.22E- 06 07	0.0006 65507 919	0.003 53115 017	3.36E- 06 -05	8.20E- 07 00737 82	0.0002 01329 1265
data set7	HTR U	1 25905 1349	0.0002 25905 1349	3.25 E-08 -05	4.60E- 05 0950	0.170 126	0.0023 63379 728	2.01E- 06 06	2.61E- 05 73	0.196 05655	2.32E- 07 73	0.0011 91905 919	0.318 00737 82	7.89E- 05 -05	0.000 42471 02536	0.0176 60377 92
averages cross datasets		2.53 E-01	0.0056 65532 059	2.34 E-05 -04	3.19E- 04 03	1.40 E-01	5.99E- 03	3.15E- 05	4.93E- 02	2.30E- 01	1.51E- 01	8.44E- 03	1.73E- 01	6.93E- 02	1.29E- 01	0.1798 04007

Drive link for excel:

<https://docs.google.com/spreadsheets/d/1VJ92cXV61h9zFSmxb3ARuxrCLnWqbHoc01KXRTCy-bU/edit?usp=sharing>

Drive link for code:

https://docs.google.com/document/d/1oTpJaKhADe68aG31k5BTp3aKbyGWr_wgygxWd2k78Pw/edit?usp=sharing

In [146]:

```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set_style('white')

from sklearn.datasets import load_digits, make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, plot_confusion_matrix, f1_score, classification_report
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler, MaxAbsScaler, StandardScaler
from sklearn.preprocessing import RobustScaler, Normalizer, QuantileTransformer, PowerTransformer
from sklearn import datasets
from sklearn import svm
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split, cross_val_score
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold, StratifiedKFold
from sklearn.linear_model import Lasso
from sklearn.pipeline import Pipeline
```

In [2]:

```
# dataset1
electrical = pd.read_csv("Data_for_UCI_named(1).csv")
electrical = electrical.replace(to_replace='unstable', value = 0)
electrical = electrical.replace(to_replace='stable', value = 1)
electricalX = electrical.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,11]]
electricalY = electrical['stabf']
```

In [25]:

```
# dataset2
AI4I = pd.read_csv("ai4i2020(1).csv")
AI4I = AI4I.replace(to_replace='M', value = 0)
AI4I = AI4I.replace(to_replace='L', value = 1)
AI4I = AI4I.replace(to_replace='H', value = 2)
AI4IX = AI4I.iloc[:,[2,3,4,5,6,7]]
AI4IY = AI4I['Machine failure']
```

In [36]:

```
#dataset3
occup_1 = pd.read_csv("datatraining(1).csv")
occup_2 = pd.read_csv("datatest(1).csv")
occup = occup_1.append(occup_2)
occupX = occup.iloc[:,[1,2,3,4,5]]
occupY = occup['Occupancy']
```

In [149]:

```
#dataset4
internet = pd.read_csv("log2.csv")
internet = internet.replace(to_replace='allow', value = 1)
internet = internet.replace(to_replace='drop', value = 0)
internet = internet.replace(to_replace='deny', value = 0)
internet = internet.replace(to_replace='reset-both', value = 0)
internetX = internet.iloc[:,[5,6,7,8,9]]
internetY = internet[ 'Action' ]
```

In [151]:

```
#dataset5
col_names = [ "AF3" , "F7" , "F3" , "FC5" , "T7" , "P7" , "O1" , "O2" , "P8" , "T8" , "FC6" ,
              "F4" , "F8" , "AF4" , "eyeDetection01" ]
eeg = pd.read_csv("EEG.csv", names=col_names)
eeg = eeg.drop([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16])
eegX = eeg.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13]]
eegY = eeg[ 'eyeDetection01' ]
```

In [141]:

```
#dataset6
credit = pd.read_excel("credit.xls", index_col=0)
credit = credit.drop('ID', axis=0)
credit = credit.values
creditX = credit[:, :23]
creditY = credit[:, 23]
```

In [153]:

```
#dataset7
htru = pd.read_csv("HTRU_2.csv", header=None)
htruX = htru.iloc[:,[0,1,2,3,4,5,6,7]]
htruY = htru.iloc[:,8]
```

Logistic Regression

In [126]:

```

pipe = Pipeline([('std',StandardScaler()),('classifier',LogisticRegression())])
search_space = [{ 'classifier': [LogisticRegression(max_iter=5000)],
                  'classifier_solver': ['saga'],
                  'classifier_penalty': ['l1', 'l2'],
                  'classifier_C': np.logspace(-4, 4, 9)},
                  {'classifier': [LogisticRegression(max_iter=5000)],
                  'classifier_solver': ['lbfgs'],
                  'classifier_penalty': ['l2'],
                  'classifier_C': np.logspace(-4, 4, 9)},
                  {'classifier': [LogisticRegression(max_iter=5000)],
                  'classifier_solver': ['lbfgs','saga'],
                  'classifier_penalty': ['none']}
                ]
]

clf = GridSearchCV(pipe, search_space, cv=StratifiedKFold(n_splits=5),
                    scoring=[ 'accuracy', 'roc_auc_ovr', 'f1_micro'], refit=False,
                    verbose=0)

```

In [154]:

```

#on dataset 1
X1_train, X1_test, Y1_train, Y1_test = train_test_split(electricalX, electricalY,
, test_size=0.33,
                                         shuffle=True,random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(electricalX, electricalY,
, test_size=0.33,
                                         shuffle=True,random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(electricalX, electricalY,
, test_size=0.33,
                                         shuffle=True,random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(electricalX, electricalY,
, test_size=0.33,
                                         shuffle=True,random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(electricalX, electricalY,
, test_size=0.33,
                                         shuffle=True,random_state=5)

```

In [156]:

```
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

Y1_pred = pipe.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
```

ROC_AUC score :0.7898624513631864
f1 score :0.7313237221494103
accuracy score: 0.8136363636363636

In [12]:

```
best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
```

ROC_AUC score :0.7926870645911238
f1 score :0.7356948228882834
accuracy score: 0.8236363636363636

In [13]:

```
best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
```

ROC_AUC score :0.7917351735358048
f1 score :0.7285006753714542
accuracy score: 0.81727272727273

In [14]:

```
best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
```

ROC_AUC score :0.8007475215861849
f1 score :0.7458834000890076
accuracy score: 0.82696969696969697

In [15]:

```
best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
```

ROC_AUC score :0.7804758665615744
f1 score :0.7210643015521064
accuracy score: 0.8093939393939394

In [158]:

```
#train set performance
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)
Y1_tpred = pipe.predict(X1_train)
print("test accuracy: " + str(accuracy_score(Y1_train, Y1_tpred)))

best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)
Y2_tpred = pipe.predict(X2_train)
print("test accuracy: " + str(accuracy_score(Y2_train, Y2_tpred)))

best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)
Y3_tpred = pipe.predict(X3_train)
print("test accuracy: " + str(accuracy_score(Y3_train, Y3_tpred)))

best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)
Y4_tpred = pipe.predict(X4_train)
print("test accuracy: " + str(accuracy_score(Y4_train, Y4_tpred)))

best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)
Y5_tpred = pipe.predict(X5_train)
print("test accuracy: " + str(accuracy_score(Y5_train, Y5_tpred)))
```

```
test accuracy: 0.8170149253731344
test accuracy: 0.8095522388059702
test accuracy: 0.815820895522388
test accuracy: 0.8113432835820895
test accuracy: 0.8182089552238806
```

In [159]:

```
#on dataset 2
X1_train, X1_test, Y1_train, Y1_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True,random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True,random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True,random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True,random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True,random_state=5)
```

In [27]:

```
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
```

ROC_AUC score :0.6085505666237155
f1 score :0.3424657534246575
accuracy score: 0.9709090909090909

In [28]:

```
best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
```

ROC_AUC score :0.6164493891201966
f1 score :0.30656934306569344
accuracy score: 0.9712121212121212

In [29]:

```
best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
```

ROC_AUC score :0.5908497703769782
f1 score :0.28776978417266186
accuracy score: 0.97

In [30]:

```
best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
```

ROC_AUC score :0.5735931758840278
f1 score :0.21705426356589147
accuracy score: 0.9693939393939394

In [31]:

```
best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
```

ROC_AUC score :0.5835112836690124
f1 score :0.22516556291390732
accuracy score: 0.9645454545454546

In [160]:

```
#train set performance
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)
Y1_tpred = pipe.predict(X1_train)
print("test accuracy: " + str(accuracy_score(Y1_train, Y1_tpred)))

best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)
Y2_tpred = pipe.predict(X2_train)
print("test accuracy: " + str(accuracy_score(Y2_train, Y2_tpred)))

best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)
Y3_tpred = pipe.predict(X3_train)
print("test accuracy: " + str(accuracy_score(Y3_train, Y3_tpred)))

best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)
Y4_tpred = pipe.predict(X4_train)
print("test accuracy: " + str(accuracy_score(Y4_train, Y4_tpred)))

best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)
Y5_tpred = pipe.predict(X5_train)
print("test accuracy: " + str(accuracy_score(Y5_train, Y5_tpred)))
```

```
test accuracy: 0.97
test accuracy: 0.9701492537313433
test accuracy: 0.9705970149253731
test accuracy: 0.9705970149253731
test accuracy: 0.9737313432835821
```

In [161]:

```
#on dataset 3
X1_train, X1_test, Y1_train, Y1_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=5)
```

In [102]:

```
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
```

ROC_AUC score :0.9872877867488372
f1 score :0.9719934102141681
accuracy score: 0.9857022708158116

In [39]:

```
best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
```

ROC_AUC score :0.985009976207944
f1 score :0.9674157303370786
accuracy score: 0.983739837398374

In [40]:

```
best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
```

ROC_AUC score :0.9856165886272178
f1 score :0.9686337054758108
accuracy score: 0.9834594897673115

In [41]:

```
best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
```

ROC_AUC score :0.9863051267212473
f1 score :0.9731543624161074
accuracy score: 0.9865433137089992

In [42]:

```
best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
```

ROC_AUC score :0.9878771363722463
f1 score :0.9778258518117902
accuracy score: 0.9885057471264368

In [162]:

```
#train set performance
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)
Y1_tpred = pipe.predict(X1_train)
print("test accuracy: " + str(accuracy_score(Y1_train, Y1_tpred)))

best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)
Y2_tpred = pipe.predict(X2_train)
print("test accuracy: " + str(accuracy_score(Y2_train, Y2_tpred)))

best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)
Y3_tpred = pipe.predict(X3_train)
print("test accuracy: " + str(accuracy_score(Y3_train, Y3_tpred)))

best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)
Y4_tpred = pipe.predict(X4_train)
print("test accuracy: " + str(accuracy_score(Y4_train, Y4_tpred)))

best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)
Y5_tpred = pipe.predict(X5_train)
print("test accuracy: " + str(accuracy_score(Y5_train, Y5_tpred)))
```

```
test accuracy: 0.9859135478525065
test accuracy: 0.9870183676287806
test accuracy: 0.9871564701008148
test accuracy: 0.9856373429084381
test accuracy: 0.9845325231321641
```

In [163]:

```
#on dataset 4
X1_train, X1_test, Y1_train, Y1_test = train_test_split(internetX, internetY, te
st_size=0.33,
                                         shuffle=True,random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(internetX, internetY, te
st_size=0.33,
                                         shuffle=True,random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(internetX, internetY, te
st_size=0.33,
                                         shuffle=True,random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(internetX, internetY, te
st_size=0.33,
                                         shuffle=True,random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(internetX, internetY, te
st_size=0.33,
                                         shuffle=True,random_state=5)
```

In [128]:

```
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
```



```
g.py:329: ConvergenceWarning: The max_iter was reached which means t  
he coef_ did not converge  
    warnings.warn("The max_iter was reached which means "  
C:\Users\dugur\anaconda3\lib\site-packages\sklearn\linear_model\sa  
g.py:329: ConvergenceWarning: The max_iter was reached which means t  
he coef_ did not converge  
    warnings.warn("The max_iter was reached which means "  
C:\Users\dugur\anaconda3\lib\site-packages\sklearn\linear_model\sa  
g.py:329: ConvergenceWarning: The max_iter was reached which means t  
he coef_ did not converge  
    warnings.warn("The max_iter was reached which means "  
  
ROC_AUC score :0.9713476325374342  
f1 score :0.9986229242608343  
accuracy score: 0.9984278183667807
```

In [129]:

```
best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
```


In [130]:

```
best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
```



```
he coef_ did not converge
warnings.warn("The max_iter was reached which means "
ROC_AUC score :0.9946369967181622
f1 score :0.9989588338939612
accuracy score: 0.9987977434569499
```

In [131]:

```
best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
```

```
ROC_AUC score :0.9942270686337396  
f1 score :0.9991978180651371  
accuracy score: 0.9990751872745769
```

In [132]:

```
best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
```



```
ROC_AUC score : 0.9944771426267838  
f1 score : 0.9987899322362053  
accuracy score: 0.9986127809118653
```

In [165]:

```
import warnings
warnings.filterwarnings('ignore')

#train set performance
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)
Y1_tpred = pipe.predict(X1_train)
print("test accuracy: " + str(accuracy_score(Y1_train, Y1_tpred)))

best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)
Y2_tpred = pipe.predict(X2_train)
print("test accuracy: " + str(accuracy_score(Y2_train, Y2_tpred)))

best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)
Y3_tpred = pipe.predict(X3_train)
print("test accuracy: " + str(accuracy_score(Y3_train, Y3_tpred)))

best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)
Y4_tpred = pipe.predict(X4_train)
print("test accuracy: " + str(accuracy_score(Y4_train, Y4_tpred)))

best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)
Y5_tpred = pipe.predict(X5_train)
print("test accuracy: " + str(accuracy_score(Y5_train, Y5_tpred)))



test accuracy: 0.9990206349929395
test accuracy: 0.9987928756889719
test accuracy: 0.9988612034801622
test accuracy: 0.9987245478977816
test accuracy: 0.9987017719673849
```

In [166]:

```
#on dataset 5
X1_train, X1_test, Y1_train, Y1_test = train_test_split(eegX, eegY, test_size=0.
33,
                                                    shuffle=True,random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(eegX, eegY, test_size=0.
33,
                                                    shuffle=True,random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(eegX, eegY, test_size=0.
33,
                                                    shuffle=True,random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(eegX, eegY, test_size=0.
33,
                                                    shuffle=True,random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(eegX, eegY, test_size=0.
33,
                                                    shuffle=True,random_state=5)
```

In [134]:

```
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
```



```
g.py:329: ConvergenceWarning: The max_iter was reached which means t  
he coef_ did not converge  
    warnings.warn("The max_iter was reached which means "  
ROC_AUC score :0.6207840543099421  
f1 score :0.544492059490799  
accuracy score: 0.6345064724919094
```

In [135]:

```
best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
```


In [136]:

```
best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
```

ROC_AUC score :0.6203768106876058
f1 score :0.5443548387096774
accuracy score: 0.6343042071197411

In [137]:

```
best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
```

ROC_AUC score :0.6275254344648502
f1 score :0.5482961824237765
accuracy score: 0.643406148867314

In [138]:

```
best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
```



```
g.py:329: ConvergenceWarning: The max_iter was reached which means t  
he coef_ did not converge  
    warnings.warn("The max_iter was reached which means "  
ROC_AUC score :0.6365178070444752  
f1 score :0.5570659143370095  
accuracy score: 0.6506877022653722
```

In [167]:

```
import warnings
warnings.filterwarnings('ignore')

#train set performance
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)
Y1_tpred = pipe.predict(X1_train)
print("test accuracy: " + str(accuracy_score(Y1_train, Y1_tpred)))

best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)
Y2_tpred = pipe.predict(X2_train)
print("test accuracy: " + str(accuracy_score(Y2_train, Y2_tpred)))

best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)
Y3_tpred = pipe.predict(X3_train)
print("test accuracy: " + str(accuracy_score(Y3_train, Y3_tpred)))

best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)
Y4_tpred = pipe.predict(X4_train)
print("test accuracy: " + str(accuracy_score(Y4_train, Y4_tpred)))

best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)
Y5_tpred = pipe.predict(X5_train)
print("test accuracy: " + str(accuracy_score(Y5_train, Y5_tpred)))
```

test accuracy: 0.6421881227580709
test accuracy: 0.6401952969310483
test accuracy: 0.6475687524910323
test accuracy: 0.6415902750099641
test accuracy: 0.6432841769629334

In [168]:

```
#on dataset 6
X1_train, X1_test, Y1_train, Y1_test = train_test_split(creditX, creditY, test_size=0.33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(creditX, creditY, test_size=0.33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(creditX, creditY, test_size=0.33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(creditX, creditY, test_size=0.33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(creditX, creditY, test_size=0.33,
                                                       shuffle=True, random_state=5)
```

In [117]:

```
best_model = clf.fit(X1_train.astype('int'), Y1_train.astype('int'))
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X1_train.astype('int'),Y1_train.astype('int'))

Y1_pred = pipe.predict(X1_test.astype('int'))
print("ROC_AUC score :" + str(roc_auc_score(Y1_test.astype('int'),Y1_pred.astype('int'))))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X1_train.astype('int'),Y1_train.astype('int'))

Y1_pred = pipe.predict(X1_test.astype('int'))
print("f1 score :" + str(f1_score(Y1_test.astype('int'),Y1_pred.astype('int'))))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train.astype('int'),Y1_train.astype('int'))

Y1_pred = pipe.predict(X1_test.astype('int'))
print("accuracy score: " + str(accuracy_score(Y1_test.astype('int'),Y1_pred.astype('int'))))

ROC_AUC score :0.5999258601109744
f1 score :0.3420689655172414
accuracy score: 0.8072727272727273
```

In [118]:

```
best_model = clf.fit(X2_train.astype('int'), Y2_train.astype('int'))
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X2_train.astype('int'),Y2_train.astype('int'))

Y2_pred = pipe.predict(X2_test.astype('int'))
print("ROC_AUC score :" + str(roc_auc_score(Y2_test.astype('int'),Y2_pred.astype('int'))))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X2_train.astype('int'),Y2_train.astype('int'))

Y2_pred = pipe.predict(X2_test.astype('int'))
print("f1 score :" + str(f1_score(Y2_test.astype('int'),Y2_pred.astype('int'))))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train.astype('int'),Y2_train.astype('int'))

Y2_pred = pipe.predict(X2_test.astype('int'))
print("accuracy score: " + str(accuracy_score(Y2_test.astype('int'),Y2_pred.astype('int'))))
```

ROC_AUC score :0.606990409395797
f1 score :0.36006884681583473
accuracy score: 0.8122222222222222

In [119]:

```
best_model = clf.fit(X3_train.astype('int'), Y3_train.astype('int'))
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X3_train.astype('int'),Y3_train.astype('int'))

Y3_pred = pipe.predict(X3_test.astype('int'))
print("ROC_AUC score :" + str(roc_auc_score(Y3_test.astype('int'),Y3_pred.astype('int'))))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X3_train.astype('int'),Y3_train.astype('int'))

Y3_pred = pipe.predict(X3_test.astype('int'))
print("f1 score :" + str(f1_score(Y3_test.astype('int'),Y3_pred.astype('int'))))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train.astype('int'),Y3_train.astype('int'))

Y3_pred = pipe.predict(X3_test.astype('int'))
print("accuracy score: " + str(accuracy_score(Y3_test.astype('int'),Y3_pred.astype('int'))))
```

ROC_AUC score :0.6074565949460267
f1 score :0.3606673766418176
accuracy score: 0.8180808080808081

In [120]:

```
best_model = clf.fit(X4_train.astype('int'), Y4_train.astype('int'))
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X4_train.astype('int'),Y4_train.astype('int'))

Y4_pred = pipe.predict(X4_test.astype('int'))
print("ROC_AUC score :" + str(roc_auc_score(Y4_test.astype('int'),Y4_pred.astype('int'))))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X4_train.astype('int'),Y4_train.astype('int'))

Y4_pred = pipe.predict(X4_test.astype('int'))
print("f1 score :" + str(f1_score(Y4_test.astype('int'),Y4_pred.astype('int'))))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train.astype('int'),Y4_train.astype('int'))

Y4_pred = pipe.predict(X4_test.astype('int'))
print("accuracy score: " + str(accuracy_score(Y4_test.astype('int'),Y4_pred.astype('int'))))
```

ROC_AUC score :0.5995760034456167
f1 score :0.3428766189502386
accuracy score: 0.8052525252525252

In [121]:

```
best_model = clf.fit(X5_train.astype('int'), Y5_train.astype('int'))
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X5_train.astype('int'),Y5_train.astype('int'))

Y5_pred = pipe.predict(X5_test.astype('int'))
print("ROC_AUC score :" + str(roc_auc_score(Y5_test.astype('int'),Y5_pred.astype('int'))))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X5_train.astype('int'),Y5_train.astype('int'))

Y5_pred = pipe.predict(X5_test.astype('int'))
print("f1 score :" + str(f1_score(Y5_test.astype('int'),Y5_pred.astype('int'))))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train.astype('int'),Y5_train.astype('int'))

Y5_pred = pipe.predict(X5_test.astype('int'))
print("accuracy score: " + str(accuracy_score(Y5_test.astype('int'),Y5_pred.astype('int'))))
```

ROC_AUC score :0.6035610274156415
f1 score :0.35157318741450067
accuracy score: 0.8084848484848485

In [169]:

```
#train set performance
best_model = clf.fit(X1_train.astype('int'), Y1_train.astype('int'))
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train.astype('int'),Y1_train.astype('int'))
Y1_tpred = pipe.predict(X1_train.astype('int'))
print("test accuracy: " + str(accuracy_score(Y1_train.astype('int')), Y1_tpred.astype('int'))))

best_model = clf.fit(X2_train.astype('int'), Y2_train.astype('int'))
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train.astype('int'),Y2_train.astype('int'))
Y2_tpred = pipe.predict(X2_train.astype('int'))
print("test accuracy: " + str(accuracy_score(Y2_train.astype('int')), Y2_tpred.astype('int'))))

best_model = clf.fit(X3_train.astype('int'), Y3_train.astype('int'))
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train.astype('int'),Y3_train.astype('int'))
Y3_tpred = pipe.predict(X3_train.astype('int'))
print("test accuracy: " + str(accuracy_score(Y3_train.astype('int')), Y3_tpred.astype('int'))))

best_model = clf.fit(X4_train.astype('int'), Y4_train.astype('int'))
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train.astype('int'),Y4_train.astype('int'))
Y4_tpred = pipe.predict(X4_train.astype('int'))
print("test accuracy: " + str(accuracy_score(Y4_train.astype('int')), Y4_tpred.astype('int'))))

best_model = clf.fit(X5_train.astype('int'), Y5_train.astype('int'))
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train.astype('int'),Y5_train.astype('int'))
Y5_tpred = pipe.predict(X5_train.astype('int'))
print("test accuracy: " + str(accuracy_score(Y5_train.astype('int')), Y5_tpred.astype('int'))))

test accuracy: 0.810497512437811
test accuracy: 0.8097512437810945
test accuracy: 0.8064179104477612
test accuracy: 0.8132835820895522
test accuracy: 0.8093532338308458
```

In [170]:

```
#on dataset 7
X1_train, X1_test, Y1_train, Y1_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True,random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True,random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True,random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True,random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True,random_state=5)
```

In [105]:

```
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)

Y1_pred = pipe.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
```

ROC_AUC score :0.916533360890564
f1 score :0.8851485148514853
accuracy score: 0.9803622820382597

In [106]:

```
best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)

Y2_pred = pipe.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
```

ROC_AUC score :0.9072324809143008
f1 score :0.8781431334622822
accuracy score: 0.97866937531742

In [107]:

```
best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)

Y3_pred = pipe.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
```

ROC_AUC score :0.9009310986964617
f1 score :0.8704253214638972
accuracy score: 0.9778229219570002

In [108]:

```
best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)

Y4_pred = pipe.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
```

ROC_AUC score :0.9078411702426241
f1 score :0.884272997032641
accuracy score: 0.9801929913661758

In [109]:

```
best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_roc_auc_ovr']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_f1_micro']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))

p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)

Y5_pred = pipe.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
```

ROC_AUC score :0.9060650728872982
f1 score :0.8762886597938144
accuracy score: 0.9796851193499239

In [171]:

```
#train set performance
best_model = clf.fit(X1_train, Y1_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X1_train,Y1_train)
Y1_tpred = pipe.predict(X1_train)
print("test accuracy: " + str(accuracy_score(Y1_train, Y1_tpred)))

best_model = clf.fit(X2_train, Y2_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X2_train,Y2_train)
Y2_tpred = pipe.predict(X2_train)
print("test accuracy: " + str(accuracy_score(Y2_train, Y2_tpred)))

best_model = clf.fit(X3_train, Y3_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X3_train,Y3_train)
Y3_tpred = pipe.predict(X3_train)
print("test accuracy: " + str(accuracy_score(Y3_train, Y3_tpred)))

best_model = clf.fit(X4_train, Y4_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X4_train,Y4_train)
Y4_tpred = pipe.predict(X4_train)
print("test accuracy: " + str(accuracy_score(Y4_train, Y4_tpred)))

best_model = clf.fit(X5_train, Y5_train)
p = best_model.cv_results_['params'][ np.argmin(best_model.cv_results_['rank_test_accuracy']) ]
pipe.set_params(**p)

pipe.fit(X5_train,Y5_train)
Y5_tpred = pipe.predict(X5_train)
print("test accuracy: " + str(accuracy_score(Y5_train, Y5_tpred)))
```

```
test accuracy: 0.9784838628971729
test accuracy: 0.9801517805020432
test accuracy: 0.9804019681427737
test accuracy: 0.978984238178634
test accuracy: 0.9789008422983905
```

In []:

In [68]:

```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set_style('white')

from sklearn.datasets import load_digits, make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, plot_confusion_matrix, f1_score, classification_report
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler, MaxAbsScaler, StandardScaler
from sklearn.preprocessing import RobustScaler, Normalizer, QuantileTransformer, PowerTransformer
from sklearn import datasets
from sklearn import svm
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split, cross_val_score
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold, StratifiedKFold
from sklearn.linear_model import Lasso
from sklearn.pipeline import Pipeline
```

In [69]:

```
# dataset1
electrical = pd.read_csv("Data_for_UCI_named(1).csv")
electrical = electrical.replace(to_replace='unstable', value = 0)
electrical = electrical.replace(to_replace='stable', value = 1)
electricalX = electrical.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,11]]
electricalY = electrical['stabf']
```

In [70]:

```
# dataset2
AI4I = pd.read_csv("ai4i2020(1).csv")
AI4I = AI4I.replace(to_replace='M', value = 0)
AI4I = AI4I.replace(to_replace='L', value = 1)
AI4I = AI4I.replace(to_replace='H', value = 2)
AI4IX = AI4I.iloc[:,[2,3,4,5,6,7]]
AI4IY = AI4I['Machine failure']
```

In [71]:

```
#dataset3
occup_1 = pd.read_csv("datatraining(1).csv")
occup_2 = pd.read_csv("datatest(1).csv")
occup = occup_1.append(occup_2)
occupX = occup.iloc[:,[1,2,3,4,5]]
occupY = occup['Occupancy']
```

In [72]:

```
#dataset4
internet = pd.read_csv("log2.csv")
internet = internet.replace(to_replace='allow', value = 1)
internet = internet.replace(to_replace='drop', value = 0)
internet = internet.replace(to_replace='deny', value = 0)
internet = internet.replace(to_replace='reset-both', value = 0)
internetX = internet.iloc[:,[5,6,7,8,9]]
internetY = internet[ 'Action' ]
```

In [85]:

```
#dataset5
col_names = [ "AF3" , "F7" , "F3" , "FC5" , "T7" , "P7" , "O1" , "O2" , "P8" , "T8" , "FC6" ,
              "F4" , "F8" , "AF4" , "eyeDetection01" ]
eeg = pd.read_csv("EEG.csv", names=col_names)
eeg = eeg.drop([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16])
eegX = eeg.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13]]
eegY = eeg[ 'eyeDetection01' ]
```

In [73]:

```
#dataset6
credit = pd.read_excel("credit.xls",index_col=0)
credit = credit.drop('ID',axis=0)
credit = credit.values
creditX = credit[:,23]
creditY = credit[:,23]
```

In [74]:

```
#dataset7
htru = pd.read_csv("HTRU_2.csv", header=None)
htruX = htru.iloc[:,[0,1,2,3,4,5,6,7]]
htruY = htru.iloc[:,8]
```

In [75]:

```
classifier=svm.SVC(kernel='rbf',probability=True)
C_list = [1,10,100,1000,10000]
gamma_list = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2]
clfacc = GridSearchCV(classifier,param_grid=[{'C': C_list,'gamma': gamma_list,
'kernel': ['rbf']}], scoring='accuracy')
clff1 = GridSearchCV(classifier,param_grid=[{'C': C_list,'gamma': gamma_list,'kernel': ['rbf']}], scoring='f1')
clfau = GridSearchCV(classifier,param_grid=[{'C': C_list,'gamma': gamma_list,'kernel': ['rbf']}], scoring='roc_auc_ovr')
```

In [98]:

```
#on dataset 1
X1_train, X1_test, Y1_train, Y1_test = train_test_split(electricalX, electricalY
, test_size=0.33,
                                         shuffle=True,random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(electricalX, electricalY
, test_size=0.33,
                                         shuffle=True,random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(electricalX, electricalY
, test_size=0.33,
                                         shuffle=True,random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(electricalX, electricalY
, test_size=0.33,
                                         shuffle=True,random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(electricalX, electricalY
, test_size=0.33,
                                         shuffle=True,random_state=5)
```

In [37]:

```
best = clfacc.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
best = clff1.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))
best = clfauc.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))
```

accuracy score: 0.9669696969696969
f1 score :0.9537547730165464
ROC_AUC score :0.9628141432803947

In [38]:

```
best = clfacc.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
best = clff1.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))
best = clfauc.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))
```

accuracy score: 0.9712121212121212
f1 score :0.9601342845153168
ROC_AUC score :0.9695823304285142

In [39]:

```
best = clfacc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
best = clff1.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))
best = clfauc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))
```

accuracy score: 0.9703030303030303
f1 score :0.958439353859203
ROC_AUC score :0.9691070287546343

In [40]:

```
best = clfacc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
best = clff1.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))
best = clfauc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))
```

accuracy score: 0.9687878787878788
f1 score :0.9560017086715079
ROC_AUC score :0.9642468819955229

In [41]:

```
best = clfacc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
best = clff1.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))
best = clfauc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))
```

accuracy score: 0.9709090909090909
f1 score :0.9590443686006825
ROC_AUC score :0.967009896191364

In [99]:

```
#train set performance
best = clfacc.fit(X1_train,Y1_train)
Y1_tpred = best.predict(X1_train)
print("accuracy score: " + str(accuracy_score(Y1_train,Y1_tpred)))

best = clfacc.fit(X2_train,Y2_train)
Y2_tpred = best.predict(X2_train)
print("accuracy score: " + str(accuracy_score(Y2_train,Y2_tpred)))

best = clfacc.fit(X3_train,Y3_train)
Y3_tpred = best.predict(X3_train)
print("accuracy score: " + str(accuracy_score(Y3_train,Y3_tpred)))

best = clfacc.fit(X4_train,Y4_train)
Y4_tpred = best.predict(X4_train)
print("accuracy score: " + str(accuracy_score(Y4_train,Y4_tpred)))

best = clfacc.fit(X5_train,Y5_train)
Y5_tpred = best.predict(X5_train)
print("accuracy score: " + str(accuracy_score(Y5_train,Y5_tpred)))
```

accuracy score: 0.9895522388059701
accuracy score: 0.9876119402985075
accuracy score: 0.9879104477611941
accuracy score: 0.991044776119403
accuracy score: 0.9976119402985074

In [100]:

```
#on dataset 2
X1_train, X1_test, Y1_train, Y1_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True,random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True,random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True,random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True,random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True,random_state=5)
```

In [43]:

```
best = clfacc.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
best = clff1.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))
best = clfauc.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))
```

accuracy score: 0.9769696969696969
f1 score :0.638095238095238
ROC_AUC score :0.7893084877919847

In [44]:

```
best = clfacc.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
best = clff1.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))
best = clfauc.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))
```

accuracy score: 0.9790909090909091
f1 score :0.6790697674418604
ROC_AUC score :0.8131526858234932

In [45]:

```
best = clfacc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
best = clff1.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))
best = clfauc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))
```

accuracy score: 0.9793939393939394
f1 score :0.6494845360824743
ROC_AUC score :0.7736478673142367

In [46]:

```
best = clfacc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
best = clff1.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))
best = clfauc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))
```

accuracy score: 0.9742424242424242
f1 score :0.6187845303867403
ROC_AUC score :0.7611766165333586

In [47]:

```
best = clfacc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
best = clff1.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))
best = clfauc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))
```

accuracy score: 0.9769696969696969
f1 score :0.6576576576576576
ROC_AUC score :0.7777723853433632

In [101]:

```
#train set performance
best = clfacc.fit(X1_train,Y1_train)
Y1_tpred = best.predict(X1_train)
print("accuracy score: " + str(accuracy_score(Y1_train,Y1_tpred)))

best = clfacc.fit(X2_train,Y2_train)
Y2_tpred = best.predict(X2_train)
print("accuracy score: " + str(accuracy_score(Y2_train,Y2_tpred)))

best = clfacc.fit(X3_train,Y3_train)
Y3_tpred = best.predict(X3_train)
print("accuracy score: " + str(accuracy_score(Y3_train,Y3_tpred)))

best = clfacc.fit(X4_train,Y4_train)
Y4_tpred = best.predict(X4_train)
print("accuracy score: " + str(accuracy_score(Y4_train,Y4_tpred)))

best = clfacc.fit(X5_train,Y5_train)
Y5_tpred = best.predict(X5_train)
print("accuracy score: " + str(accuracy_score(Y5_train,Y5_tpred)))
```

accuracy score: 0.9892537313432835
accuracy score: 0.9895522388059701
accuracy score: 0.9905970149253731
accuracy score: 0.9782089552238806
accuracy score: 0.9905970149253731

In [102]:

```
#on dataset 3
X1_train, X1_test, Y1_train, Y1_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=5)
```

In [49]:

```
best = clfacc.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
best = clff1.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))
best = clfauc.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))

accuracy score: 0.9865433137089992
f1 score :0.9733629300776914
ROC_AUC score :0.9884126461414131
```

In [52]:

```
best = clfacc.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
best = clff1.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))
best = clfauc.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))

accuracy score: 0.9848612279226241
f1 score :0.9694224235560589
ROC_AUC score :0.9869330102778939
```

In [53]:

```
best = clfacc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
best = clff1.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))
best = clfauc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))
```

accuracy score: 0.984300532660499
f1 score :0.9697297297297297
ROC_AUC score :0.9876530099693094

In [54]:

```
best = clfacc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
best = clff1.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))
best = clfauc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))
```

accuracy score: 0.9893467900196243
f1 score :0.9784335981838821
ROC_AUC score :0.9881604142908207

In [55]:

```
best = clfacc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
best = clff1.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))
best = clfauc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))
```

accuracy score: 0.9854219231847491
f1 score :0.9713656387665198
ROC_AUC score :0.9874812657226696

In [103]:

```
#train set performance
best = clfacc.fit(X1_train,Y1_train)
Y1_tpred = best.predict(X1_train)
print("accuracy score: " + str(accuracy_score(Y1_train,Y1_tpred)))

best = clfacc.fit(X2_train,Y2_train)
Y2_tpred = best.predict(X2_train)
print("accuracy score: " + str(accuracy_score(Y2_train,Y2_tpred)))

best = clfacc.fit(X3_train,Y3_train)
Y3_tpred = best.predict(X3_train)
print("accuracy score: " + str(accuracy_score(Y3_train,Y3_tpred)))

best = clfacc.fit(X4_train,Y4_train)
Y4_tpred = best.predict(X4_train)
print("accuracy score: " + str(accuracy_score(Y4_train,Y4_tpred)))

best = clfacc.fit(X5_train,Y5_train)
Y5_tpred = best.predict(X5_train)
print("accuracy score: " + str(accuracy_score(Y5_train,Y5_tpred)))
```

accuracy score: 0.9915757492059107
 accuracy score: 0.9908852368457396
 accuracy score: 0.9929567739262533
 accuracy score: 0.991713851677945
 accuracy score: 0.9935091838143902

In [104]:

```
#on dataset 4
X1_train, X1_test, Y1_train, Y1_test = train_test_split(internetX, internetY, train_size=5000,
                                                       shuffle=True,random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(internetX, internetY, train_size=5000,
                                                       shuffle=True,random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(internetX, internetY, train_size=5000,
                                                       shuffle=True,random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(internetX, internetY, train_size=5000,
                                                       shuffle=True,random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(internetX, internetY, train_size=5000,
                                                       shuffle=True,random_state=5)
```

In [79]:

```
best = clfacc.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
best = clff1.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))
best = clfauc.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))
```

accuracy score: 0.9972576488468909
f1 score :0.9980194040989723
ROC_AUC score :0.9955914398525015

In [80]:

```
best = clfacc.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
best = clff1.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))
best = clfauc.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))
```

accuracy score: 0.9979514967290029
f1 score :0.998216340621404
ROC_AUC score :0.9947032113578929

In [81]:

```
best = clfacc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
best = clff1.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))
best = clfauc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))
```

accuracy score: 0.9988931474261548
f1 score :0.9990372318259546
ROC_AUC score :0.9947953040851497

In [82]:

```
best = clfacc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
best = clff1.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))
best = clfauc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))
```

accuracy score: 0.9984966629220908
f1 score :0.9986931483635632
ROC_AUC score :0.9947564034219558

In [83]:

```
best = clfacc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
best = clff1.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))
best = clfauc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))
```

accuracy score: 0.9987609859248001
f1 score :0.9989199775355326
ROC_AUC score :0.9948108842440792

In [105]:

```
#train set performance
best = clfacc.fit(X1_train,Y1_train)
Y1_tpred = best.predict(X1_train)
print("accuracy score: " + str(accuracy_score(Y1_train,Y1_tpred)))

best = clfacc.fit(X2_train,Y2_train)
Y2_tpred = best.predict(X2_train)
print("accuracy score: " + str(accuracy_score(Y2_train,Y2_tpred)))

best = clfacc.fit(X3_train,Y3_train)
Y3_tpred = best.predict(X3_train)
print("accuracy score: " + str(accuracy_score(Y3_train,Y3_tpred)))

best = clfacc.fit(X4_train,Y4_train)
Y4_tpred = best.predict(X4_train)
print("accuracy score: " + str(accuracy_score(Y4_train,Y4_tpred)))

best = clfacc.fit(X5_train,Y5_train)
Y5_tpred = best.predict(X5_train)
print("accuracy score: " + str(accuracy_score(Y5_train,Y5_tpred)))
```

accuracy score: 0.9996
accuracy score: 0.9998
accuracy score: 0.9992
accuracy score: 0.9992
accuracy score: 0.9992

In [106]:

```
#on dataset 5
X1_train, X1_test, Y1_train, Y1_test = train_test_split(eegX, eegY, test_size=0.33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(eegX, eegY, test_size=0.33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(eegX, eegY, test_size=0.33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(eegX, eegY, test_size=0.33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(eegX, eegY, test_size=0.33,
                                                       shuffle=True, random_state=5)
```

In [87]:

```
best = clfacc.fit(X1_train, Y1_train)
Y1_pred = best.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test, Y1_pred)))
best = clff1.fit(X1_train, Y1_train)
Y1_pred = best.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test, Y1_pred)))
best = clfauc.fit(X1_train, Y1_train)
Y1_pred = best.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test, Y1_pred)))

accuracy score: 0.9797734627831716
f1 score :0.9775078722447144
ROC_AUC score :0.979726713826792
```

In [88]:

```
best = clfacc.fit(X2_train, Y2_train)
Y2_pred = best.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test, Y2_pred)))
best = clff1.fit(X2_train, Y2_train)
Y2_pred = best.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test, Y2_pred)))
best = clfauc.fit(X2_train, Y2_train)
Y2_pred = best.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test, Y2_pred)))

accuracy score: 0.9743122977346278
f1 score :0.9715692858741886
ROC_AUC score :0.9739061642383423
```

In [89]:

```
best = clfacc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
best = clff1.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))
best = clfauc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))
```

accuracy score: 0.9749190938511327
f1 score :0.9719710669077758
ROC_AUC score :0.9746795411228459

In [90]:

```
best = clfacc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
best = clff1.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))
best = clfauc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))
```

accuracy score: 0.9765372168284789
f1 score :0.9734675205855443
ROC_AUC score :0.9760848459423002

In [91]:

```
best = clfacc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
best = clff1.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))
best = clfauc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))
```

accuracy score: 0.9755258899676376
f1 score :0.9727784026996625
ROC_AUC score :0.975535825825508

In [107]:

```
#train set performance
best = clfacc.fit(X1_train,Y1_train)
Y1_tpred = best.predict(X1_train)
print("accuracy score: " + str(accuracy_score(Y1_train,Y1_tpred)))

best = clfacc.fit(X2_train,Y2_train)
Y2_tpred = best.predict(X2_train)
print("accuracy score: " + str(accuracy_score(Y2_train,Y2_tpred)))

best = clfacc.fit(X3_train,Y3_train)
Y3_tpred = best.predict(X3_train)
print("accuracy score: " + str(accuracy_score(Y3_train,Y3_tpred)))

best = clfacc.fit(X4_train,Y4_train)
Y4_tpred = best.predict(X4_train)
print("accuracy score: " + str(accuracy_score(Y4_train,Y4_tpred)))

best = clfacc.fit(X5_train,Y5_train)
Y5_tpred = best.predict(X5_train)
print("accuracy score: " + str(accuracy_score(Y5_train,Y5_tpred)))
```

accuracy score: 0.9983060980470307
 accuracy score: 0.9986050219210841
 accuracy score: 0.9980071741729772
 accuracy score: 0.9989039457951375
 accuracy score: 0.9986050219210841

In [108]:

```
#on dataset 6
X1_train, X1_test, Y1_train, Y1_test = train_test_split(creditX, creditY, train_
size=5000,
                                         shuffle=True,random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(creditX, creditY, train_
size=5000,
                                         shuffle=True,random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(creditX, creditY, train_
size=5000,
                                         shuffle=True,random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(creditX, creditY, train_
size=5000,
                                         shuffle=True,random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(creditX, creditY, train_
size=5000,
                                         shuffle=True,random_state=5)
```

In [93]:

```
best = clfacc.fit(X1_train.astype('int'),Y1_train.astype('int'))
Y1_pred = best.predict(X1_test.astype('int'))
print("accuracy score: " + str(accuracy_score(Y1_test.astype('int')),Y1_pred.astype('int'))))
best = clff1.fit(X1_train.astype('int'),Y1_train.astype('int'))
Y1_pred = best.predict(X1_test.astype('int'))
print("f1 score :" + str(f1_score(Y1_test.astype('int')),Y1_pred.astype('int'))))
best = clfauc.fit(X1_train.astype('int'),Y1_train.astype('int'))
Y1_pred = best.predict(X1_test.astype('int'))
print("ROC_AUC score :" + str(roc_auc_score(Y1_test.astype('int')),Y1_pred.astype('int'))))
```

accuracy score: 0.77696
f1 score :0.07708570499268175
ROC_AUC score :0.5110725917881075

In [94]:

```
best = clfacc.fit(X2_train.astype('int'),Y2_train.astype('int'))
Y2_pred = best.predict(X2_test.astype('int'))
print("accuracy score: " + str(accuracy_score(Y2_test.astype('int')),Y2_pred.astype('int'))))
best = clff1.fit(X2_train.astype('int'),Y2_train.astype('int'))
Y2_pred = best.predict(X2_test.astype('int'))
print("f1 score :" + str(f1_score(Y2_test.astype('int')),Y2_pred.astype('int'))))
best = clfauc.fit(X2_train.astype('int'),Y2_train.astype('int'))
Y2_pred = best.predict(X2_test.astype('int'))
print("ROC_AUC score :" + str(roc_auc_score(Y2_test.astype('int')),Y2_pred.astype('int'))))
```

accuracy score: 0.77928
f1 score :0.03339711254131153
ROC_AUC score :0.5038629491081827

In [95]:

```
best = clfacc.fit(X3_train.astype('int'),Y3_train.astype('int'))
Y3_pred = best.predict(X3_test.astype('int'))
print("accuracy score: " + str(accuracy_score(Y3_test.astype('int')),Y3_pred.astype('int'))))
best = clff1.fit(X3_train.astype('int'),Y3_train.astype('int'))
Y3_pred = best.predict(X3_test.astype('int'))
print("f1 score :" + str(f1_score(Y3_test.astype('int')),Y3_pred.astype('int'))))
best = clfauc.fit(X3_train.astype('int'),Y3_train.astype('int'))
Y3_pred = best.predict(X3_test.astype('int'))
print("ROC_AUC score :" + str(roc_auc_score(Y3_test.astype('int')),Y3_pred.astype('int'))))
```

accuracy score: 0.77768
f1 score :0.06658974781605406
ROC_AUC score :0.509419969126538

In [96]:

```
best = clfacc.fit(X4_train.astype('int'),Y4_train.astype('int'))
Y4_pred = best.predict(X4_test.astype('int'))
print("accuracy score: " + str(accuracy_score(Y4_test.astype('int'),Y4_pred.astype('int'))))
best = clff1.fit(X4_train.astype('int'),Y4_train.astype('int'))
Y4_pred = best.predict(X4_test.astype('int'))
print("f1 score :" + str(f1_score(Y4_test.astype('int'),Y4_pred.astype('int'))))
best = clfauc.fit(X4_train.astype('int'),Y4_train.astype('int'))
Y4_pred = best.predict(X4_test.astype('int'))
print("ROC_AUC score :" + str(roc_auc_score(Y4_test.astype('int'),Y4_pred.astype('int'))))
```

accuracy score: 0.77992
f1 score :0.061728395061728385
ROC_AUC score :0.5080069346339311

In [97]:

```
best = clfacc.fit(X5_train.astype('int'),Y5_train.astype('int'))
Y5_pred = best.predict(X5_test.astype('int'))
print("accuracy score: " + str(accuracy_score(Y5_test.astype('int'),Y5_pred.astype('int'))))
best = clff1.fit(X5_train.astype('int'),Y5_train.astype('int'))
Y5_pred = best.predict(X5_test.astype('int'))
print("f1 score :" + str(f1_score(Y5_test.astype('int'),Y5_pred.astype('int'))))
best = clfauc.fit(X5_train.astype('int'),Y5_train.astype('int'))
Y5_pred = best.predict(X5_test.astype('int'))
print("ROC_AUC score :" + str(roc_auc_score(Y5_test.astype('int'),Y5_pred.astype('int'))))
```

accuracy score: 0.7764
f1 score :0.07255571823653816
ROC_AUC score :0.5095367434548889

In [109]:

```
#train set performance
best = clfacc.fit(X1_train.astype('int'),Y1_train.astype('int'))
Y1_tpred = best.predict(X1_train.astype('int'))
print("accuracy score: " + str(accuracy_score(Y1_train.astype('int'),Y1_tpred.astype('int'))))

best = clfacc.fit(X2_train.astype('int'),Y2_train.astype('int'))
Y2_tpred = best.predict(X2_train.astype('int'))
print("accuracy score: " + str(accuracy_score(Y2_train.astype('int'),Y2_tpred.astype('int'))))

best = clfacc.fit(X3_train.astype('int'),Y3_train.astype('int'))
Y3_tpred = best.predict(X3_train.astype('int'))
print("accuracy score: " + str(accuracy_score(Y3_train.astype('int'),Y3_tpred.astype('int'))))

best = clfacc.fit(X4_train.astype('int'),Y4_train.astype('int'))
Y4_tpred = best.predict(X4_train.astype('int'))
print("accuracy score: " + str(accuracy_score(Y4_train.astype('int'),Y4_tpred.astype('int'))))

best = clfacc.fit(X5_train.astype('int'),Y5_train.astype('int'))
Y5_tpred = best.predict(X5_train.astype('int'))
print("accuracy score: " + str(accuracy_score(Y5_train.astype('int'),Y5_tpred.astype('int'))))
```

accuracy score: 0.9976
accuracy score: 0.9844
accuracy score: 0.9988
accuracy score: 0.991
accuracy score: 0.9868

In [110]:

```
#on dataset 7
X1_train, X1_test, Y1_train, Y1_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True, random_state=5)
```

In [59]:

```
best = clfacc.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("accuracy score: " + str(accuracy_score(Y1_test,Y1_pred)))
best = clff1.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("f1 score :" + str(f1_score(Y1_test,Y1_pred)))
best = clfauc.fit(X1_train,Y1_train)
Y1_pred = best.predict(X1_test)
print("ROC_AUC score :" + str(roc_auc_score(Y1_test,Y1_pred)))
```

accuracy score: 0.9778229219570002
f1 score :0.8678102926337032
ROC_AUC score :0.8994540456411978

In [60]:

```
best = clfacc.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("accuracy score: " + str(accuracy_score(Y2_test,Y2_pred)))
best = clff1.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("f1 score :" + str(f1_score(Y2_test,Y2_pred)))
best = clfauc.fit(X2_train,Y2_train)
Y2_pred = best.predict(X2_test)
print("ROC_AUC score :" + str(roc_auc_score(Y2_test,Y2_pred)))
```

accuracy score: 0.9800237006940917
f1 score :0.8856589147286822
ROC_AUC score :0.9039220957395527

In [61]:

```
best = clfacc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("accuracy score: " + str(accuracy_score(Y3_test,Y3_pred)))
best = clff1.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("f1 score :" + str(f1_score(Y3_test,Y3_pred)))
best = clfauc.fit(X3_train,Y3_train)
Y3_pred = best.predict(X3_test)
print("ROC_AUC score :" + str(roc_auc_score(Y3_test,Y3_pred)))
```

accuracy score: 0.978838665989504
f1 score :0.876604146100691
ROC_AUC score :0.8918994413407821

In [62]:

```
best = clfacc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("accuracy score: " + str(accuracy_score(Y4_test,Y4_pred)))
best = clff1.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("f1 score :" + str(f1_score(Y4_test,Y4_pred)))
best = clfauc.fit(X4_train,Y4_train)
Y4_pred = best.predict(X4_test)
print("ROC_AUC score :" + str(roc_auc_score(Y4_test,Y4_pred)))
```

```
accuracy score: 0.9796851193499239
f1 score :0.8726919339164237
ROC_AUC score :0.9122260717451655
```

In [63]:

```
best = clfacc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("accuracy score: " + str(accuracy_score(Y5_test,Y5_pred)))
best = clff1.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("f1 score :" + str(f1_score(Y5_test,Y5_pred)))
best = clfauc.fit(X5_train,Y5_train)
Y5_pred = best.predict(X5_test)
print("ROC_AUC score :" + str(roc_auc_score(Y5_test,Y5_pred)))
```

```
accuracy score: 0.9803622820382597
f1 score :0.8809034907597536
ROC_AUC score :0.8963029596920581
```

In [111]:

```
#train set performance
best = clfacc.fit(X1_train,Y1_train)
Y1_tpred = best.predict(X1_train)
print("accuracy score: " + str(accuracy_score(Y1_train,Y1_tpred)))

best = clfacc.fit(X2_train,Y2_train)
Y2_tpred = best.predict(X2_train)
print("accuracy score: " + str(accuracy_score(Y2_train,Y2_tpred)))

best = clfacc.fit(X3_train,Y3_train)
Y3_tpred = best.predict(X3_train)
print("accuracy score: " + str(accuracy_score(Y3_train,Y3_tpred)))

best = clfacc.fit(X4_train,Y4_train)
Y4_tpred = best.predict(X4_train)
print("accuracy score: " + str(accuracy_score(Y4_train,Y4_tpred)))

best = clfacc.fit(X5_train,Y5_train)
Y5_tpred = best.predict(X5_train)
print("accuracy score: " + str(accuracy_score(Y5_train,Y5_tpred)))
```

accuracy score: 0.9839045951130014
accuracy score: 0.9816529063464264
accuracy score: 0.9838211992327579
accuracy score: 0.9822366775081312
accuracy score: 0.9810691351847218

In []:

In [1]:

```
from numpy import arange
import numpy as np
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, RepeatedKFold
from sklearn.linear_model import Lasso, LogisticRegression
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import roc_auc_score
```

In [2]:

```
# all datasets no need cleaning
dataset1_df = pd.read_csv("Data_for_UCI_named.csv")
dataset1 = dataset1_df.values

dataset2_df = pd.read_csv("ai4i2020.csv")
dataset2 = dataset2_df.values

dataset3_df_part1 = pd.read_csv("datatraining.csv")
dataset3_df_part2 = pd.read_csv("datatest.csv")
dataset3_df = dataset3_df_part1.append(dataset3_df_part2)
dataset3 = dataset3_df.values

dataset4_df = pd.read_csv("log2.csv")
dataset4 = dataset4_df.values

col_names = ["AF3", "F7", "F3", "FC5", "T7", "P7", "O1", "O2", "P8", "T8", "FC6", "F4",
             "F8", "AF4", "eyeDetection01"]
dataset5_df = pd.read_csv("EEGEyeState.csv", names=col_names)
dataset5_df = dataset5_df.drop([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16])
dataset5 = dataset5_df.values

dataset6_df = pd.read_excel("default of credit card clients.xls", index_col = 0)
dataset6_df = dataset6_df.drop('ID', axis = 0)
dataset6 = dataset6_df.values

dataset7_df = pd.read_csv("HTRU_2.csv", header=None)
dataset7 = dataset7_df.values
```

In [3]:

```
# overview for dataset variables

##### convert label to number dataset 1, no need cleaning #####
# 11 predictive attributes, 1 non-predictive(p1), 2 goal fields. exclude nonpredictive
data1 = dataset1[:,[0,1,2,3,5,6,7,8,9,10,11,12,13]]
data1_X = data1[:,12]
data1_Y_temp = data1[:,12]
data1_Y = []
# convert the categorical variable
for index, value in enumerate(data1_Y_temp):
    if value == 'stable': # stable
        data1_Y.append(1)
    elif value == 'unstable': # unstable
        data1_Y.append(0)
data1_Y = np.array(data1_Y)
data1_Y = data1_Y[:, None]
# revised dataset, convert categorical variable to number, more clear
data1 = np.append(data1_X,data1_Y, axis = 1)
print('Dataset1:')
print(data1.shape)
print(data1_X.shape)
print(data1_Y.shape)

##### convert categorical variable to number dataset 2, no need cleaning #####
data2 = dataset2[:,[2,3,4,5,6,7,8,9,10,11,12,13]]
# convert the categorical variable
for index, row in enumerate(data2):
    if row[0] == 'M': # M
        row[0] = 0
    elif row[0] == 'L': # L
        row[0] = 1
    else:
        row[0] = 2 # H

data2_X = data2[:,[0,1,2,3,4,5,7,8,9,10,11]]
data2_Y = data2[:,6]
data2_Y = np.array(data2_Y)
data2_Y = data2_Y[:, None]
print('Dataset2:')
print(dataset2_df['Type'].unique())
print(data2.shape)
print(data2_X.shape)
print(data2_Y.shape)

##### dataset 3, no need cleaning #####
data3 = dataset3[:,[1,2,3,4,5,6]]
data3_X = data3[:,[0,1,2,3,4]]
data3_Y = data3[:,5]
data3_Y = np.array(data3_Y)
data3_Y = data3_Y[:, None]
print('Dataset3:')
print(data3.shape)
print(data3_X.shape)
print(data3_Y.shape)

##### convert label to number dataset 4, no need cleaning #####

```

```
#####
data4 = dataset4
data4_X = data1[:,[0,1,2,3,5,6,7,8,9,10,11]]
data4_Y_temp = data1[:,4]
data4_Y = []
# convert the categorical variable
for index, value in enumerate(data4_Y_temp):
    if value == 'allow' : # allow
        data4_Y.append(1)
    else: # not allow
        data4_Y.append(0)
data4_Y = np.array(data4_Y)
data4_Y = data4_Y[:, None]
# revised dataset, convert categorical variable to number, more clear
data4 = np.append(data4_X,data4_Y, axis = 1)
print('Dataset4:')
print(data4.shape)
print(data4_X.shape)
print(data4_Y.shape)

##### dataset 5, no need cleaning #####
data5 = dataset5
data5_X = data5[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13]]
data5_Y = data5[:,14]
data5_Y = np.array(data5_Y)
data5_Y = data5_Y[:, None]
print('Dataset5:')
print(data5.shape)
print(data5_X.shape)
print(data5_Y.shape)

##### dataset 6, no need cleaning #####
data6 = dataset6
data6_X = data6[:,23]
data6_Y = data6[:,23]
data6_Y = np.array(data6_Y)
data6_Y = data6_Y[:, None]
print('Dataset6:')
print(data6.shape)
print(data6_X.shape)
print(data6_Y.shape)

##### dataset 7, no need cleaning #####
data7 = dataset7
data7_X = data7[:,8]
data7_Y = data7[:,8]
data7_Y = np.array(data7_Y)
data7_Y = data7_Y[:, None]
print('Dataset7:')
print(data7.shape)
print(data7_X.shape)
print(data7_Y.shape)
```

```
Dataset1:  
(10000, 13)  
(10000, 12)  
(10000, 1)  
Dataset2:  
[ 'M' 'L' 'H' ]  
(10000, 12)  
(10000, 11)  
(10000, 1)  
Dataset3:  
(10808, 6)  
(10808, 5)  
(10808, 1)  
Dataset4:  
(10000, 12)  
(10000, 11)  
(10000, 1)  
Dataset5:  
(14980, 15)  
(14980, 14)  
(14980, 1)  
Dataset6:  
(30000, 24)  
(30000, 23)  
(30000, 1)  
Dataset7:  
(17898, 9)  
(17898, 8)  
(17898, 1)
```

In [4]:

```
# set shuffle to True so that we avoid just choosing the first n data. Instead we want the training and test data to
# distribute throughout the dataset.
# set random state so that each of us will reproduce consistent training and test data everytime we run it.

# create 5 trials, each with new splitted training and test set data
data1_train_X = []
data1_train_Y = []
data1_test_X = []
data1_test_Y = []
data1_train = []
data1_test = [] # create 6 lists in order to put in all train&test set splitted at each trial

data2_train_X = []
data2_train_Y = []
data2_test_X = []
data2_test_Y = []
data2_train = []
data2_test = []

data3_train_X = []
data3_train_Y = []
data3_test_X = []
data3_test_Y = []
data3_train = []
data3_test = []

data4_train_X = []
data4_train_Y = []
data4_test_X = []
data4_test_Y = []
data4_train = []
data4_test = []

data5_train_X = []
data5_train_Y = []
data5_test_X = []
data5_test_Y = []
data5_train = []
data5_test = []

data6_train_X = []
data6_train_Y = []
data6_test_X = []
data6_test_Y = []
data6_train = []
data6_test = []

data7_train_X = []
data7_train_Y = []
data7_test_X = []
data7_test_Y = []
data7_train = []
data7_test = []

##### dataset1 split 5 trials#####
```

```

for i in range(5):
    train1, test1 = train_test_split(data1, test_size=0.33, shuffle = True, random_
_state = (1+i))
    data1_train.append(train1)
    data1_test.append(test1)
    data1_train_X.append(train1[:, :12])
    data1_train_Y.append(train1[:, 12][:, None])
    data1_test_X.append(test1[:, :12])
    data1_test_Y.append(test1[:, 12][:, None])
data1_train = np.array(data1_train)
data1_test = np.array(data1_test)
data1_train_X = np.array(data1_train_X)
data1_train_Y = np.array(data1_train_Y)
data1_test_X = np.array(data1_test_X)
data1_test_Y = np.array(data1_test_Y)
print('Dataset1:')
print(data1_train.shape)
print(data1_test.shape)
print(data1_train_X.shape)
print(data1_train_Y.shape)
print(data1_test_X.shape)
print(data1_test_Y.shape)

##### dataset2 split 5 trials#####
for i in range(5):
    train2, test2 = train_test_split(data2, test_size=0.33, shuffle = True, random_
_state = (1+i))
    data2_train.append(train2)
    data2_test.append(test2)
    data2_train_X.append(train2[:, [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11]])
    data2_train_Y.append(train2[:, 6][:, None])
    data2_test_X.append(test2[:, [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11]])
    data2_test_Y.append(test2[:, 6][:, None])
data2_train = np.array(data2_train)
data2_test = np.array(data2_test)
data2_train_X = np.array(data2_train_X)
data2_train_Y = np.array(data2_train_Y)
data2_test_X = np.array(data2_test_X)
data2_test_Y = np.array(data2_test_Y)
print('Dataset2:')
print(data2_train.shape)
print(data2_test.shape)
print(data2_train_X.shape)
print(data2_train_Y.shape)
print(data2_test_X.shape)
print(data2_test_Y.shape)

##### dataset3 split 5 trials#####
for i in range(5):
    train3, test3 = train_test_split(data3, test_size=0.33, shuffle = True, random_
_state = (1+i))
    data3_train.append(train3)
    data3_test.append(test3)
    data3_train_X.append(train3[:, [0, 1, 2, 3, 4]])
    data3_train_Y.append(train3[:, 5][:, None])
    data3_test_X.append(test3[:, [0, 1, 2, 3, 4]])
    data3_test_Y.append(test3[:, 5][:, None])
data3_train = np.array(data3_train)
data3_test = np.array(data3_test)
data3_train_X = np.array(data3_train_X)
data3_train_Y = np.array(data3_train_Y)

```

```

data3_test_X = np.array(data3_test_X)
data3_test_Y = np.array(data3_test_Y)
print('Dataset3:')
print(data3_train.shape)
print(data3_test.shape)
print(data3_train_X.shape)
print(data3_train_Y.shape)
print(data3_test_X.shape)
print(data3_test_Y.shape)

##### dataset4 split 5 trials#####
for i in range(5):
    train4, test4 = train_test_split(data4, test_size=0.33, shuffle = True, random_state = (1+i))
    data4_train.append(train4)
    data4_test.append(test4)
    data4_train_X.append(train4[:,[0,1,2,3,5,6,7,8,9,10,11]])
    data4_train_Y.append(train4[:,4][:,None])
    data4_test_X.append(test4[:,[0,1,2,3,5,6,7,8,9,10,11]])
    data4_test_Y.append(test4[:,4][:,None])
data4_train = np.array(data4_train)
data4_test = np.array(data4_test)
data4_train_X = np.array(data4_train_X)
data4_train_Y = np.array(data4_train_Y)
data4_test_X = np.array(data4_test_X)
data4_test_Y = np.array(data4_test_Y)
print('Dataset4:')
print(data4_train.shape)
print(data4_test.shape)
print(data4_train_X.shape)
print(data4_train_Y.shape)
print(data4_test_X.shape)
print(data4_test_Y.shape)

##### dataset5 split 5 trials#####
for i in range(5):
    train5, test5 = train_test_split(data5, test_size=0.33, shuffle = True, random_state = (1+i))
    data5_train.append(train5)
    data5_test.append(test5)
    data5_train_X.append(train5[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13]])
    data5_train_Y.append(train5[:,14][:,None])
    data5_test_X.append(test5[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13]])
    data5_test_Y.append(test5[:,14][:,None])
data5_train = np.array(data5_train)
data5_test = np.array(data5_test)
data5_train_X = np.array(data5_train_X)
data5_train_Y = np.array(data5_train_Y)
data5_test_X = np.array(data5_test_X)
data5_test_Y = np.array(data5_test_Y)
print('Dataset5:')
print(data5_train.shape)
print(data5_test.shape)
print(data5_train_X.shape)
print(data5_train_Y.shape)
print(data5_test_X.shape)
print(data5_test_Y.shape)

##### dataset6 split 5 trials#####
for i in range(5):
    train6, test6 = train_test_split(data6, test_size=0.33, shuffle = True, random_

```

```
_state = (1+i))
data6_train.append(train6)
data6_test.append(test6)
data6_train_X.append(train6[:, :23])
data6_train_Y.append(train6[:, 23][:, None])
data6_test_X.append(test6[:, :23])
data6_test_Y.append(test6[:, 23][:, None])
data6_train = np.array(data6_train)
data6_test = np.array(data6_test)
data6_train_X = np.array(data6_train_X)
data6_train_Y = np.array(data6_train_Y)
data6_test_X = np.array(data6_test_X)
data6_test_Y = np.array(data6_test_Y)
print('Dataset6:')
print(data6_train.shape)
print(data6_test.shape)
print(data6_train_X.shape)
print(data6_train_Y.shape)
print(data6_test_X.shape)
print(data6_test_Y.shape)

##### dataset7 split 5 trials#####
for i in range(5):
    train7, test7 = train_test_split(data7, test_size=0.33, shuffle = True, random_
_state = (1+i))
    data7_train.append(train7)
    data7_test.append(test7)
    data7_train_X.append(train7[:, :8])
    data7_train_Y.append(train7[:, 8][:, None])
    data7_test_X.append(test7[:, :8])
    data7_test_Y.append(test7[:, 8][:, None])
data7_train = np.array(data7_train)
data7_test = np.array(data7_test)
data7_train_X = np.array(data7_train_X)
data7_train_Y = np.array(data7_train_Y)
data7_test_X = np.array(data7_test_X)
data7_test_Y = np.array(data7_test_Y)
print('Dataset7:')
print(data7_train.shape)
print(data7_test.shape)
print(data7_train_X.shape)
print(data7_train_Y.shape)
print(data7_test_X.shape)
print(data7_test_Y.shape)
```

```
Dataset1:  
(5, 6700, 13)  
(5, 3300, 13)  
(5, 6700, 12)  
(5, 6700, 1)  
(5, 3300, 12)  
(5, 3300, 1)  
Dataset2:  
(5, 6700, 12)  
(5, 3300, 12)  
(5, 6700, 11)  
(5, 6700, 1)  
(5, 3300, 11)  
(5, 3300, 1)  
Dataset3:  
(5, 7241, 6)  
(5, 3567, 6)  
(5, 7241, 5)  
(5, 7241, 1)  
(5, 3567, 5)  
(5, 3567, 1)  
Dataset4:  
(5, 6700, 12)  
(5, 3300, 12)  
(5, 6700, 11)  
(5, 6700, 1)  
(5, 3300, 11)  
(5, 3300, 1)  
Dataset5:  
(5, 10036, 15)  
(5, 4944, 15)  
(5, 10036, 14)  
(5, 10036, 1)  
(5, 4944, 14)  
(5, 4944, 1)  
Dataset6:  
(5, 20100, 24)  
(5, 9900, 24)  
(5, 20100, 23)  
(5, 20100, 1)  
(5, 9900, 23)  
(5, 9900, 1)  
Dataset7:  
(5, 11991, 9)  
(5, 5907, 9)  
(5, 11991, 8)  
(5, 11991, 1)  
(5, 5907, 8)  
(5, 5907, 1)
```

In [5]:

```
##### K-Nearest Neighbors ##### block0 recall  
the required packages  
from random import seed  
from random import randrange  
from math import sqrt  
%config InlineBackend.figure_format = 'retina'  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn import datasets  
import scipy  
from matplotlib.colors import ListedColormap  
from functools import partial  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.model_selection import GridSearchCV  
from sklearn.preprocessing import StandardScaler  
from sklearn.metrics import classification_report,confusion_matrix, accuracy_sco  
re
```

In [6]:

```
#####
# K-Nearest Neighbors #####
#### block1 normalization #####
standardScaler = StandardScaler()
#####
# data1 #####
knn_data1_train_X = []
knn_data1_test_X = []
for i in range(5):
    standardScaler.fit(data1_train_X[i])
    standardScaler.mean_
    standardScaler.scale_
    # use transform to normalize
    knn_data1_train_X_temp = standardScaler.transform(data1_train_X[i])
    knn_data1_train_X.append(knn_data1_train_X_temp)
    knn_data1_test_X_temp = standardScaler.transform(data1_test_X[i])
    knn_data1_test_X.append(knn_data1_test_X_temp)

knn_data2_train_X = []
knn_data2_test_X = []
#####
# data2 #####
for i in range(5):
    standardScaler.fit(data2_train_X[i])
    standardScaler.mean_
    standardScaler.scale_
    # use transform to normalize
    knn_data2_train_X_temp = standardScaler.transform(data2_train_X[i])
    knn_data2_train_X.append(knn_data2_train_X_temp)
    knn_data2_test_X_temp = standardScaler.transform(data2_test_X[i])
    knn_data2_test_X.append(knn_data2_test_X_temp)

knn_data3_train_X = []
knn_data3_test_X = []
#####
# data3 #####
for i in range(5):
    standardScaler.fit(data3_train_X[i])
    standardScaler.mean_
    standardScaler.scale_
    # use transform to normalize
    knn_data3_train_X_temp = standardScaler.transform(data3_train_X[i])
    knn_data3_train_X.append(knn_data3_train_X_temp)
    knn_data3_test_X_temp = standardScaler.transform(data3_test_X[i])
    knn_data3_test_X.append(knn_data3_test_X_temp)

knn_data4_train_X = []
knn_data4_test_X = []
#####
# data4 #####
for i in range(5):
    standardScaler.fit(data4_train_X[i])
    standardScaler.mean_
    standardScaler.scale_
    # use transform to normalize
    knn_data4_train_X_temp = standardScaler.transform(data4_train_X[i])
    knn_data4_train_X.append(knn_data4_train_X_temp)
    knn_data4_test_X_temp = standardScaler.transform(data4_test_X[i])
    knn_data4_test_X.append(knn_data4_test_X_temp)

knn_data5_train_X = []
knn_data5_test_X = []
#####
# data5 #####
for i in range(5):
```

```
standardScaler.fit(data5_train_X[i])
standardScaler.mean_
standardScaler.scale_
# use transform to normalize
knn_data5_train_X_temp = standardScaler.transform(data5_train_X[i])
knn_data5_train_X.append(knn_data5_train_X_temp)
knn_data5_test_X_temp = standardScaler.transform(data5_test_X[i])
knn_data5_test_X.append(knn_data5_test_X_temp)

knn_data6_train_X = []
knn_data6_test_X = []
##### data6 #####
for i in range(5):
    standardScaler.fit(data6_train_X[i])
    standardScaler.mean_
    standardScaler.scale_
    # use transform to normalize
    knn_data6_train_X_temp = standardScaler.transform(data6_train_X[i])
    knn_data6_train_X.append(knn_data6_train_X_temp)
    knn_data6_test_X_temp = standardScaler.transform(data6_test_X[i])
    knn_data6_test_X.append(knn_data6_test_X_temp)

knn_data7_train_X = []
knn_data7_test_X = []
##### data7 #####
for i in range(5):
    standardScaler.fit(data7_train_X[i])
    standardScaler.mean_
    standardScaler.scale_
    # use transform to normalize
    knn_data7_train_X_temp = standardScaler.transform(data7_train_X[i])
    knn_data7_train_X.append(knn_data7_train_X_temp)
    knn_data7_test_X_temp = standardScaler.transform(data7_test_X[i])
    knn_data7_test_X.append(knn_data7_test_X_temp)
```

In [7]:

```
#####
# K-Nearest Neighbors #####
#### block2 #####
knn_clf = KNeighborsClassifier()
# Cross validate model with Kfold cross val
kfold = KFold(n_splits=5, random_state= None, shuffle=False)
k_range=[i for i in range(1,11)]

param_grid = [
    {
        'weights': ['uniform'],
        'n_neighbors': k_range
    },
    {
        'weights': ['distance'],
        'n_neighbors': k_range,
        'p':[i for i in range(1,6)]
    }
]

grid_search = GridSearchCV(knn_clf, param_grid = param_grid, cv=kfold,n_jobs=-1, verbose=2)
#####
# use grid_search to train data #####
print('Trial 1')
grid_search.fit(knn_data1_train_X[0],data1_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data1_test_X[0].astype('int')) # fit test set
knn_clf.score(knn_data1_test_X[0], y_pred_t1)
```

Trial 1
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=9, p=1, weights='distance')
best_score
0.956865671641791
best_parameters
{'n_neighbors': 9, 'p': 1, 'weights': 'distance'}

Out[7]:

0.850909090909091

In [8]:

```
print('confusion_matrix for trial 1: ')
print(confusion_matrix(data1_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1: ')
print(classification_report(data1_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1: ')
print(accuracy_score(data1_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data1_test_Y[0].astype('int'),y_pred_t1)))
```

confusion_matrix for trial 1:

```
[[2109    4]
 [ 532  655]]
```

classification_report for trial 1:

	precision	recall	f1-score	support
0	0.80	1.00	0.89	2113
1	0.99	0.55	0.71	1187
accuracy			0.84	3300
macro avg	0.90	0.77	0.80	3300
weighted avg	0.87	0.84	0.82	3300

accuracy_score for trial 1:

```
0.8375757575757575
```

```
ROC_AUC score :0.7749591229485222
```

In [9]:

```
#####
# K-Nearest Neighbors #####
#### use grid_search to train data #####
print('Trial 2')
grid_search.fit(knn_data1_train_X[1],data1_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data1_test_X[1].astype('int')) # fit test set
knn_clf.score(knn_data1_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data1_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data1_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data1_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data1_test_Y[1].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
print('Trial 3')
grid_search.fit(knn_data1_train_X[2],data1_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data1_test_X[2].astype('int')) # fit test set
knn_clf.score(knn_data1_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data1_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data1_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
print(accuracy_score(data1_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data1_test_Y[2].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
print('Trial 4')
grid_search.fit(knn_data1_train_X[3],data1_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data1_test_X[3].astype('int')) # fit test set
knn_clf.score(knn_data1_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data1_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data1_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data1_test_Y[3].astype('int'),y_pred_t1))
```

```
print("ROC_AUC score :" + str(roc_auc_score(data1_test_Y[3].astype('int'),y_pred_t1)))
##### use grid_search to train data #####
print('Trial 5')
grid_search.fit(knn_data1_train_X[4],data1_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data1_test_X[4].astype('int')) # fit test set
knn_clf.score(knn_data1_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data1_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data1_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data1_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data1_test_Y[4].astype('int'),y_pred_t1)))
```

```

Trial 2
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=10, p=1, weights='distance')
best_score
0.9547761194029849
best_parameters
{'n_neighbors': 10, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 2:
[[2110    3]
 [ 539  648]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0         0.80      1.00      0.89     2113
          1         1.00      0.55      0.71     1187

   accuracy                           0.84      3300
  macro avg         0.90      0.77      0.80      3300
weighted avg        0.87      0.84      0.82      3300

accuracy_score for trial 2:
0.8357575757575758
ROC_AUC score :0.7722471433908357
Trial 3
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=10, p=1, weights='distance')
best_score
0.9541791044776119
best_parameters
{'n_neighbors': 10, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 3:
[[2122    7]
 [ 513  658]]
classification_report for trial 3:
      precision    recall   f1-score   support
          0         0.81      1.00      0.89     2129
          1         0.99      0.56      0.72     1171

   accuracy                           0.84      3300
  macro avg         0.90      0.78      0.80      3300
weighted avg        0.87      0.84      0.83      3300

accuracy_score for trial 3:
0.8424242424242424
ROC_AUC score :0.7793124831782963
Trial 4
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=10, p=1, weights='distance')
best_score
0.956268656716418
best_parameters
{'n_neighbors': 10, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 4:
[[2117    3]
 [ 551  629]]
classification_report for trial 4:
      precision    recall   f1-score   support

```

```

          0      0.79      1.00      0.88      2120
          1      1.00      0.53      0.69      1180

accuracy
macro avg      0.89      0.77      0.79      3300
weighted avg    0.87      0.83      0.82      3300

accuracy_score for trial 4:
0.8321212121212122
ROC_AUC score :0.7658178765590022
Trial 5
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=10, p=1, weights='distance')
best_score
0.9592537313432835
best_parameters
{'n_neighbors': 10, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 5:
[[2119   2]
 [ 540  639]]
classification_report for trial 5:
      precision    recall    f1-score   support

          0      0.80      1.00      0.89      2121
          1      1.00      0.54      0.70      1179

accuracy
macro avg      0.90      0.77      0.79      3300
weighted avg    0.87      0.84      0.82      3300

accuracy_score for trial 5:
0.8357575757575758
ROC_AUC score :0.7705208906932134

```

In [10]:

```
#####
# K-Nearest Neighbors #####
#### use grid_search to train data #####
print('Trial 1')
grid_search.fit(knn_data2_train_X[0],data2_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data2_test_X[0].astype('int')) # fit test set
knn_clf.score(knn_data2_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data2_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data2_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data2_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data2_test_Y[0].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
print('Trial 2')
grid_search.fit(knn_data2_train_X[1],data2_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data2_test_X[1].astype('int')) # fit test set
knn_clf.score(knn_data2_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data2_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data2_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data2_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data2_test_Y[1].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
print('Trial 3')
grid_search.fit(knn_data2_train_X[2],data2_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data2_test_X[2].astype('int')) # fit test set
knn_clf.score(knn_data2_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data2_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data2_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
print(accuracy_score(data2_test_Y[2].astype('int'),y_pred_t1))
```

```
print("ROC_AUC score :" + str(roc_auc_score(data2_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data #####
print('Trial 4')
grid_search.fit(knn_data2_train_X[3],data2_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data2_test_X[3].astype('int')) # fit test set
knn_clf.score(knn_data2_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data2_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data2_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data2_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data2_test_Y[3].astype('int'),y_pred_t1)))
#####
print('Trial 5')
grid_search.fit(knn_data2_train_X[4],data2_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data2_test_X[4].astype('int')) # fit test set
knn_clf.score(knn_data2_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data2_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data2_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data2_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data2_test_Y[4].astype('int'),y_pred_t1)))
```

```

Trial 1
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=2)
best_score
0.9988059701492537
best_parameters
{'n_neighbors': 2, 'weights': 'uniform'}
confusion_matrix for trial 1:
[[3186    0]
 [ 2   112]]
classification_report for trial 1:
      precision    recall  f1-score   support

          0       1.00     1.00     1.00      3186
          1       1.00     0.98     0.99      114

   accuracy                           1.00      3300
 macro avg       1.00     0.99     1.00      3300
weighted avg     1.00     1.00     1.00      3300

accuracy_score for trial 1:
0.9993939393939394
ROC_AUC score :0.9912280701754386
Trial 2
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=2)
best_score
0.9986567164179105
best_parameters
{'n_neighbors': 2, 'weights': 'uniform'}
confusion_matrix for trial 2:
[[3185    0]
 [ 1   114]]
classification_report for trial 2:
      precision    recall  f1-score   support

          0       1.00     1.00     1.00      3185
          1       1.00     0.99     1.00      115

   accuracy                           1.00      3300
 macro avg       1.00     1.00     1.00      3300
weighted avg     1.00     1.00     1.00      3300

accuracy_score for trial 2:
0.9996969696969698
ROC_AUC score :0.9956521739130435
Trial 3
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=2)
best_score
0.998955223880597
best_parameters
{'n_neighbors': 2, 'weights': 'uniform'}
confusion_matrix for trial 3:
[[3186    0]
 [ 3   111]]
classification_report for trial 3:
      precision    recall  f1-score   support
```

```

          0      1.00      1.00      1.00      3186
          1      1.00      0.97      0.99      114

accuracy
macro avg      1.00      0.99      0.99      3300
weighted avg    1.00      1.00      1.00      3300

accuracy_score for trial 3:
0.9990909090909091
ROC_AUC score :0.986842105263158
Trial 4
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=2)
best_score
0.9992537313432835
best_parameters
{'n_neighbors': 2, 'weights': 'uniform'}
confusion_matrix for trial 4:
[[3194    0]
 [   5  101]]
classification_report for trial 4:
      precision    recall    f1-score   support
          0      1.00      1.00      1.00      3194
          1      1.00      0.95      0.98      106

accuracy
macro avg      1.00      0.98      0.99      3300
weighted avg    1.00      1.00      1.00      3300

accuracy_score for trial 4:
0.9984848484848485
ROC_AUC score :0.9764150943396226
Trial 5
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=2)
best_score
0.998955223880597
best_parameters
{'n_neighbors': 2, 'weights': 'uniform'}
confusion_matrix for trial 5:
[[3170    0]
 [   3  127]]
classification_report for trial 5:
      precision    recall    f1-score   support
          0      1.00      1.00      1.00      3170
          1      1.00      0.98      0.99      130

accuracy
macro avg      1.00      0.99      0.99      3300
weighted avg    1.00      1.00      1.00      3300

accuracy_score for trial 5:
0.9990909090909091
ROC_AUC score :0.9884615384615385

```

In [11]:

```
#####
# K-Nearest Neighbors #####
##### use grid_search to train data #####
# data3 trial1 #####
print('Trial 1')
grid_search.fit(knn_data3_train_X[0],data3_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data3_test_X[0].astype('int')) # fit test set
knn_clf.score(knn_data3_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data3_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data3_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data3_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data3_test_Y[0].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
# data3 trial2 #####
print('Trial 2')
grid_search.fit(knn_data3_train_X[1],data3_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data3_test_X[1].astype('int')) # fit test set
knn_clf.score(knn_data3_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data3_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data3_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data3_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data3_test_Y[1].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
# data3 trial3 #####
print('Trial 3')
grid_search.fit(knn_data3_train_X[2],data3_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data3_test_X[2].astype('int')) # fit test set
knn_clf.score(knn_data3_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data3_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data3_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
```

```

print(accuracy_score(data3_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data3_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data ##### data3 trial4 #####
print('Trial 4')
grid_search.fit(knn_data3_train_X[3],data3_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data3_test_X[3].astype('int')) # fit test set
knn_clf.score(knn_data3_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data3_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data3_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data3_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data3_test_Y[3].astype('int'),y_pred_t1)))
#####
use grid_search to train data ##### data3 trial5 #####
print('Trial 5')
grid_search.fit(knn_data3_train_X[4],data3_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data3_test_X[4].astype('int')) # fit test set
knn_clf.score(knn_data3_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data3_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data3_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data3_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data3_test_Y[4].astype('int'),y_pred_t1)))

```

```

Trial 1
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=1)
best_score
0.9925424850058528
best_parameters
{'n_neighbors': 1, 'weights': 'uniform'}
confusion_matrix for trial 1:
[[2611  67]
 [ 52 837]]
classification_report for trial 1:
      precision    recall   f1-score   support
          0       0.98      0.97      0.98      2678
          1       0.93      0.94      0.93      889

   accuracy                           0.97      3567
  macro avg       0.95      0.96      0.96      3567
weighted avg       0.97      0.97      0.97      3567

accuracy_score for trial 1:
0.9666386319035604
ROC_AUC score :0.9582443204681566
Trial 2
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=3, p=1, weights='distance')
best_score
0.9929566590027796
best_parameters
{'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 2:
[[2619  85]
 [ 13 850]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0       1.00      0.97      0.98      2704
          1       0.91      0.98      0.95      863

   accuracy                           0.97      3567
  macro avg       0.95      0.98      0.96      3567
weighted avg       0.97      0.97      0.97      3567

accuracy_score for trial 2:
0.9725259321558732
ROC_AUC score :0.9767506787935303
Trial 3
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=3, weights='distance')
best_score
0.9917141370119991
best_parameters
{'n_neighbors': 3, 'p': 2, 'weights': 'distance'}
confusion_matrix for trial 3:
[[2587  66]
 [ 79 835]]
classification_report for trial 3:
      precision    recall   f1-score   support

```

```

0      0.97      0.98      0.97      2653
1      0.93      0.91      0.92      914

accuracy          0.96      3567
macro avg        0.95      0.94      0.95      3567
weighted avg     0.96      0.96      0.96      3567

accuracy_score for trial 3:
0.959349593495935
ROC_AUC score :0.9443446212165576
Trial 4
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=4, p=5, weights='distance')
best_score
0.9924043634588914
best_parameters
{'n_neighbors': 4, 'p': 5, 'weights': 'distance'}
confusion_matrix for trial 4:
[[2639  56]
 [ 11 861]]
classification_report for trial 4:
      precision    recall   f1-score   support
0         1.00      0.98      0.99      2695
1         0.94      0.99      0.96      872

accuracy          0.98      3567
macro avg        0.97      0.98      0.98      3567
weighted avg     0.98      0.98      0.98      3567

accuracy_score for trial 4:
0.9812167087188113
ROC_AUC score :0.9833030501608484
Trial 5
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(p=3, weights='distance')
best_score
0.9918516866270888
best_parameters
{'n_neighbors': 5, 'p': 3, 'weights': 'distance'}
confusion_matrix for trial 5:
[[2609  54]
 [  6 898]]
classification_report for trial 5:
      precision    recall   f1-score   support
0         1.00      0.98      0.99      2663
1         0.94      0.99      0.97      904

accuracy          0.98      3567
macro avg        0.97      0.99      0.98      3567
weighted avg     0.98      0.98      0.98      3567

accuracy_score for trial 5:
0.9831791421362489
ROC_AUC score :0.9865424748852681

```

In [12]:

```
#####
# K-Nearest Neighbors #####
#### use grid_search to train data #####
# data4 trial1 #####
print('Trial 1')
grid_search.fit(knn_data4_train_X[0],data4_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data4_test_X[0].astype('int')) # fit test set
knn_clf.score(knn_data4_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data4_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data4_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data4_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data4_test_Y[0].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
# data4 trial2 #####
print('Trial 2')
grid_search.fit(knn_data4_train_X[1],data4_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data4_test_X[1].astype('int')) # fit test set
knn_clf.score(knn_data4_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data4_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data4_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data4_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data4_test_Y[1].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
# data4 trial3 #####
print('Trial 3')
grid_search.fit(knn_data4_train_X[2],data4_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data4_test_X[2].astype('int')) # fit test set
knn_clf.score(knn_data4_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data4_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data4_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
```

```

print(accuracy_score(data4_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data4_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data ##### data4 trial4 #####
print('Trial 4')
grid_search.fit(knn_data4_train_X[3],data4_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data4_test_X[3].astype('int')) # fit test set
knn_clf.score(knn_data4_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data4_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data4_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data4_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data4_test_Y[3].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data ##### data4 trial5 #####
print('Trial 5')
grid_search.fit(knn_data4_train_X[4],data4_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data4_test_X[4].astype('int')) # fit test set
knn_clf.score(knn_data4_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data4_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data4_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data4_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data4_test_Y[4].astype('int'),y_pred_t1)))

```

```

Trial 1
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=8)
best_score
0.6461194029850746
best_parameters
{'n_neighbors': 8, 'weights': 'uniform'}
confusion_matrix for trial 1:
[[2026 145]
 [1047  82]]
classification_report for trial 1:
      precision    recall   f1-score   support
          -1       0.66     0.93     0.77     2171
           0       0.36     0.07     0.12     1129

   accuracy                           0.64     3300
  macro avg       0.51     0.50     0.45     3300
weighted avg       0.56     0.64     0.55     3300

accuracy_score for trial 1:
0.6387878787878788
ROC_AUC score :0.50292057433134
Trial 2
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=8)
best_score
0.643134328358209
best_parameters
{'n_neighbors': 8, 'weights': 'uniform'}
confusion_matrix for trial 2:
[[2024 181]
 [ 997  98]]
classification_report for trial 2:
      precision    recall   f1-score   support
          -1       0.67     0.92     0.77     2205
           0       0.35     0.09     0.14     1095

   accuracy                           0.64     3300
  macro avg       0.51     0.50     0.46     3300
weighted avg       0.56     0.64     0.56     3300

accuracy_score for trial 2:
0.6430303030303031
ROC_AUC score :0.5037057745472617
Trial 3
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=10)
best_score
0.6529850746268657
best_parameters
{'n_neighbors': 10, 'weights': 'uniform'}
confusion_matrix for trial 3:
[[2030 107]
 [1099  64]]
classification_report for trial 3:
      precision    recall   f1-score   support

```

```

-1      0.65      0.95      0.77      2137
      0      0.37      0.06      0.10      1163

accuracy          0.63      3300
macro avg        0.51      0.50      0.43      3300
weighted avg     0.55      0.63      0.53      3300

accuracy_score for trial 3:
0.6345454545454545
ROC_AUC score :0.5024799513626153
Trial 4
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=10)
best_score
0.6446268656716418
best_parameters
{'n_neighbors': 10, 'weights': 'uniform'}
confusion_matrix for trial 4:
[[2048 140]
 [1055  57]]
classification_report for trial 4:
      precision    recall   f1-score   support
-1      0.66      0.94      0.77      2188
      0      0.29      0.05      0.09      1112

accuracy          0.64      3300
macro avg        0.47      0.49      0.43      3300
weighted avg     0.54      0.64      0.54      3300

accuracy_score for trial 4:
0.6378787878787879
ROC_AUC score :0.4936368090171373
Trial 5
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=10)
best_score
0.6495522388059701
best_parameters
{'n_neighbors': 10, 'weights': 'uniform'}
confusion_matrix for trial 5:
[[2014 139]
 [1084  63]]
classification_report for trial 5:
      precision    recall   f1-score   support
-1      0.65      0.94      0.77      2153
      0      0.31      0.05      0.09      1147

accuracy          0.63      3300
macro avg        0.48      0.50      0.43      3300
weighted avg     0.53      0.63      0.53      3300

accuracy_score for trial 5:
0.62939393939394
ROC_AUC score :0.49518240803469216

```

In [13]:

```
#####
# K-Nearest Neighbors #####
#### use grid_search to train data #####
print('Trial 1')
grid_search.fit(knn_data5_train_X[0],data5_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data5_test_X[0].astype('int')) # fit test set
knn_clf.score(knn_data5_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data5_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data5_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data5_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data5_test_Y[0].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
print('Trial 2')
grid_search.fit(knn_data5_train_X[1],data5_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data5_test_X[1].astype('int')) # fit test set
knn_clf.score(knn_data5_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data5_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data5_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data5_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data5_test_Y[1].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
print('Trial 3')
grid_search.fit(knn_data5_train_X[2],data5_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data5_test_X[2].astype('int')) # fit test set
knn_clf.score(knn_data5_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data5_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data5_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
print(accuracy_score(data5_test_Y[2].astype('int'),y_pred_t1))
```

```
print("ROC_AUC score :" + str(roc_auc_score(data5_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data #####
print('Trial 4')
grid_search.fit(knn_data5_train_X[3],data5_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data5_test_X[3].astype('int')) # fit test set
knn_clf.score(knn_data5_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data5_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data5_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data5_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data5_test_Y[3].astype('int'),y_pred_t1)))
#####
print('Trial 5')
grid_search.fit(knn_data5_train_X[4],data5_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data5_test_X[4].astype('int')) # fit test set
knn_clf.score(knn_data5_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data5_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data5_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data5_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data5_test_Y[4].astype('int'),y_pred_t1)))
```

```

Trial 1
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=4, p=1, weights='distance')
best_score
0.9067353406503533
best_parameters
{'n_neighbors': 4, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 1:
[[2534 191]
 [2008 211]]
classification_report for trial 1:
      precision    recall   f1-score   support
          0       0.56      0.93      0.70      2725
          1       0.52      0.10      0.16      2219

   accuracy                           0.56      4944
  macro avg       0.54      0.51      0.43      4944
weighted avg       0.54      0.56      0.46      4944

accuracy_score for trial 1:
0.5552184466019418
ROC_AUC score :0.5124980671514981
Trial 2
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=6, p=1, weights='distance')
best_score
0.8346967883324699
best_parameters
{'n_neighbors': 6, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 2:
[[ 261 2445]
 [ 126 2112]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0       0.67      0.10      0.17      2706
          1       0.46      0.94      0.62      2238

   accuracy                           0.48      4944
  macro avg       0.57      0.52      0.40      4944
weighted avg       0.58      0.48      0.37      4944

accuracy_score for trial 2:
0.4799757281553398
ROC_AUC score :0.5200760300315652
Trial 3
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=6, p=1, weights='distance')
best_score
0.8699679111158753
best_parameters
{'n_neighbors': 6, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 3:
[[ 302 2431]
 [ 167 2044]]
classification_report for trial 3:
      precision    recall   f1-score   support

```

```

          0      0.64      0.11      0.19      2733
          1      0.46      0.92      0.61      2211

accuracy
macro avg      0.55      0.52      0.40      4944
weighted avg     0.56      0.47      0.38      4944

accuracy_score for trial 3:
0.47451456310679613
ROC_AUC score :0.517484923451796
Trial 4
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=4, p=1, weights='distance')
best_score
0.9191915447328771
best_parameters
{'n_neighbors': 4, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 4:
[[ 947 1808]
 [ 537 1652]]
classification_report for trial 4:
      precision    recall   f1-score   support
          0      0.64      0.34      0.45      2755
          1      0.48      0.75      0.58      2189

accuracy
macro avg      0.56      0.55      0.52      4944
weighted avg     0.57      0.53      0.51      4944

accuracy_score for trial 4:
0.5256877022653722
ROC_AUC score :0.5492105802067588
Trial 5
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=6, p=1, weights='distance')
best_score
0.8277205081021206
best_parameters
{'n_neighbors': 6, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 5:
[[2634   94]
 [2082  134]]
classification_report for trial 5:
      precision    recall   f1-score   support
          0      0.56      0.97      0.71      2728
          1      0.59      0.06      0.11      2216

accuracy
macro avg      0.57      0.51      0.41      4944
weighted avg     0.57      0.56      0.44      4944

accuracy_score for trial 5:
0.5598705501618123
ROC_AUC score :0.5130059180367786

```

In [14]:

```
#####
# K-Nearest Neighbors #####
#### block8 #####
knn_clf = KNeighborsClassifier(algorithm = "kd_tree", leaf_size = 400)
# Cross validate model with Kfold cross val
kfold = KFold(n_splits=5, random_state= None, shuffle=False)
k_range=[i for i in range(1,11)]

param_grid = [
    {
        'weights': ['uniform'],
        'n_neighbors': k_range
    },
    {
        'weights': ['distance'],
        'n_neighbors': k_range,
        'p':[i for i in [1,2]]
    }
]

grid_search = GridSearchCV(knn_clf, param_grid = param_grid, cv=kfold,n_jobs=-1, verbose=2)
#####
# use grid_search to train data #####
data6_trial1 #####
print('Trial 1')
grid_search.fit(knn_data6_train_X[0],data6_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data6_test_X[0].astype('int')) # fit test set
knn_clf.score(knn_data6_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data6_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data6_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data6_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data6_test_Y[0].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
data6_trial2 #####
print('Trial 2')
grid_search.fit(knn_data6_train_X[1],data6_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data6_test_X[1].astype('int')) # fit test set
knn_clf.score(knn_data6_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data6_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data6_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data6_test_Y[1].astype('int'),y_pred_t1))
```

```

print("ROC_AUC score :" + str(roc_auc_score(data6_test_Y[1].astype('int'),y_pred_t1)))
##### use grid_search to train data ##### data6 trial3 #####
print('Trial 3')
grid_search.fit(knn_data6_train_X[2],data6_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data6_test_X[2].astype('int')) # fit test set
knn_clf.score(knn_data6_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data6_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data6_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
print(accuracy_score(data6_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data6_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data ##### data6 trial4 #####
print('Trial 4')
grid_search.fit(knn_data6_train_X[3],data6_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data6_test_X[3].astype('int')) # fit test set
knn_clf.score(knn_data6_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data6_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data6_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data6_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data6_test_Y[3].astype('int'),y_pred_t1)))
##### use grid_search to train data ##### data6 trials5 #####
print('Trial 5')
grid_search.fit(knn_data6_train_X[4],data6_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data6_test_X[4].astype('int')) # fit test set
knn_clf.score(knn_data6_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data6_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data6_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data6_test_Y[4].astype('int'),y_pred_t1))

```

```
print("ROC_AUC score :" + str(roc_auc_score(data6_test_Y[4].astype('int'),y_pred_t1)))
```

```

Trial 1
Fitting 5 folds for each of 30 candidates, totalling 150 fits
best_estimator
KNeighborsClassifier(algorithm='kd_tree', leaf_size=400, n_neighbors
=10)
best_score
0.8064179104477611
best_parameters
{'n_neighbors': 10, 'weights': 'uniform'}
confusion_matrix for trial 1:
[[7611  79]
 [2030  180]]
classification_report for trial 1:
      precision    recall   f1-score   support
          0       0.79      0.99      0.88     7690
          1       0.69      0.08      0.15     2210

   accuracy                           0.79      9900
  macro avg       0.74      0.54      0.51      9900
weighted avg       0.77      0.79      0.71      9900

accuracy_score for trial 1:
0.7869696969696969
ROC_AUC score :0.5355874409381638
Trial 2
Fitting 5 folds for each of 30 candidates, totalling 150 fits
best_estimator
KNeighborsClassifier(algorithm='kd_tree', leaf_size=400, n_neighbors
=10)
best_score
0.8065671641791043
best_parameters
{'n_neighbors': 10, 'weights': 'uniform'}
confusion_matrix for trial 2:
[[7599  107]
 [1892  302]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0       0.80      0.99      0.88     7706
          1       0.74      0.14      0.23     2194

   accuracy                           0.80      9900
  macro avg       0.77      0.56      0.56      9900
weighted avg       0.79      0.80      0.74      9900

accuracy_score for trial 2:
0.79808080808081
ROC_AUC score :0.5618814235364789
Trial 3
Fitting 5 folds for each of 30 candidates, totalling 150 fits
best_estimator
KNeighborsClassifier(algorithm='kd_tree', leaf_size=400, n_neighbors
=10)
best_score
0.8030348258706468
best_parameters
{'n_neighbors': 10, 'weights': 'uniform'}
confusion_matrix for trial 3:
[[7718   68]]

```

```
[1945 169]]
classification_report for trial 3:
      precision    recall   f1-score   support
      0         0.80     0.99     0.88     7786
      1         0.71     0.08     0.14     2114

      accuracy          0.80     9900
      macro avg       0.76     0.54     0.51     9900
      weighted avg    0.78     0.80     0.73     9900

accuracy_score for trial 3:
0.7966666666666666
ROC_AUC score :0.535604805559113
Trial 4
Fitting 5 folds for each of 30 candidates, totalling 150 fits
best_estimator
KNeighborsClassifier(algorithm='kd_tree', leaf_size=400, n_neighbors=10)
best_score
0.8081094527363184
best_parameters
{'n_neighbors': 10, 'weights': 'uniform'}
confusion_matrix for trial 4:
[[7611  67]
 [2076 146]]
classification_report for trial 4:
      precision    recall   f1-score   support
      0         0.79     0.99     0.88     7678
      1         0.69     0.07     0.12     2222

      accuracy          0.78     9900
      macro avg       0.74     0.53     0.50     9900
      weighted avg    0.76     0.78     0.71     9900

accuracy_score for trial 4:
0.7835353535353535
ROC_AUC score :0.5284901699338989
Trial 5
Fitting 5 folds for each of 30 candidates, totalling 150 fits
best_estimator
KNeighborsClassifier(algorithm='kd_tree', leaf_size=400, n_neighbors=10)
best_score
0.8056716417910448
best_parameters
{'n_neighbors': 10, 'weights': 'uniform'}
confusion_matrix for trial 5:
[[7569  97]
 [1971 263]]
classification_report for trial 5:
      precision    recall   f1-score   support
      0         0.79     0.99     0.88     7666
      1         0.73     0.12     0.20     2234

      accuracy          0.79     9900
      macro avg       0.76     0.55     0.54     9900
      weighted avg    0.78     0.79     0.73     9900
```

```
accuracy_score for trial 5:  
0.791111111111111  
ROC_AUC score :0.5525363888635211
```

In [15]:

```
#####
# K-Nearest Neighbors #####
#### block9 #####
knn_clf = KNeighborsClassifier()
# Cross validate model with Kfold cross val
kfold = KFold(n_splits=5, random_state= None, shuffle=False)
k_range=[i for i in range(1,11)]

param_grid = [
    {
        'weights':['uniform'],
        'n_neighbors': k_range
    },
    {
        'weights':['distance'],
        'n_neighbors': k_range,
        'p':[i for i in range(1,6)]
    }
]

grid_search = GridSearchCV(knn_clf, param_grid = param_grid, cv=kfold,n_jobs=-1, verbose=2)
#####
# use grid_search to train data #####
data7_trial1 #####
print('Trial 1')
grid_search.fit(knn_data7_train_X[0],data7_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data7_test_X[0].astype('int')) # fit test set
knn_clf.score(knn_data7_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data7_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data7_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data7_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data7_test_Y[0].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
data7_trial2 #####
print('Trial 2')
grid_search.fit(knn_data7_train_X[1],data7_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data7_test_X[1].astype('int')) # fit test set
knn_clf.score(knn_data7_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data7_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data7_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data7_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data7_test_Y[1].astype('int'),y_pred_t1)))
```

```

_t1)))
#####
# use grid_search to train data #####
data7_trial3 #####
print('Trial 3')
grid_search.fit(knn_data7_train_X[2],data7_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data7_test_X[2].astype('int')) # fit test set
knn_clf.score(knn_data7_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data7_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data7_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
print(accuracy_score(data7_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data7_test_Y[2].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
data7_trial4 #####
print('Trial 4')
grid_search.fit(knn_data7_train_X[3],data7_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data7_test_X[3].astype('int')) # fit test set
knn_clf.score(knn_data7_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data7_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data7_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data7_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data7_test_Y[3].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
data7_trial5 #####
print('Trial 5')
grid_search.fit(knn_data7_train_X[4],data7_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
knn_clf = grid_search.best_estimator_
y_pred_t1 = knn_clf.predict(knn_data7_test_X[4].astype('int')) # fit test set
knn_clf.score(knn_data7_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data7_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data7_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data7_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data7_test_Y[4].astype('int'),y_pred_t1)))

```



```

Trial 1
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(weights='distance')
best_score
0.9792343626636202
best_parameters
{'n_neighbors': 5, 'p': 2, 'weights': 'distance'}
confusion_matrix for trial 1:
[[5362  12]
 [ 146 387]]
classification_report for trial 1:
      precision    recall   f1-score   support
          0       0.97     1.00     0.99     5374
          1       0.97     0.73     0.83     533
accuracy                           0.97    5907
macro avg       0.97     0.86     0.91    5907
weighted avg    0.97     0.97     0.97    5907

accuracy_score for trial 1:
0.973252073810733
ROC_AUC score :0.8619229128365259
Trial 2
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(p=1, weights='distance')
best_score
0.9794847449990458
best_parameters
{'n_neighbors': 5, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 2:
[[5346  18]
 [ 161 382]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0       0.97     1.00     0.98     5364
          1       0.95     0.70     0.81     543
accuracy                           0.97    5907
macro avg       0.96     0.85     0.90    5907
weighted avg    0.97     0.97     0.97    5907

accuracy_score for trial 2:
0.9696969696969697
ROC_AUC score :0.8500716872458501
Trial 3
Fitting 5 folds for each of 60 candidates, totalling 300 fits
best_estimator
KNeighborsClassifier(n_neighbors=8, p=1, weights='distance')
best_score
0.9786508904704178
best_parameters
{'n_neighbors': 8, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 3:
[[5356  14]
 [ 155 382]]
classification_report for trial 3:
      precision    recall   f1-score   support

```

	precision	recall	f1-score	support
0	0.97	1.00	0.98	5370
1	0.96	0.71	0.82	537
accuracy			0.97	5907
macro avg	0.97	0.85	0.90	5907
weighted avg	0.97	0.97	0.97	5907

accuracy_score for trial 3:
0.9713898764178094
ROC_AUC score :0.8543761638733706

Trial 4

Fitting 5 folds for each of 60 candidates, totalling 300 fits

best_estimator
KNeighborsClassifier(n_neighbors=8, p=4, weights='distance')
best_score
0.9789011337431741
best_parameters
{'n_neighbors': 8, 'p': 4, 'weights': 'distance'}
confusion_matrix for trial 4:
[[5357 14]
 [140 396]]

classification_report for trial 4:

	precision	recall	f1-score	support
0	0.97	1.00	0.99	5371
1	0.97	0.74	0.84	536
accuracy			0.97	5907
macro avg	0.97	0.87	0.91	5907
weighted avg	0.97	0.97	0.97	5907

accuracy_score for trial 4:
0.9739292364990689
ROC_AUC score :0.868099689598924

Trial 5

Fitting 5 folds for each of 60 candidates, totalling 300 fits

best_estimator
KNeighborsClassifier(p=1, weights='distance')
best_score
0.9789843975162016
best_parameters
{'n_neighbors': 5, 'p': 1, 'weights': 'distance'}
confusion_matrix for trial 5:
[[5364 24]
 [149 370]]

classification_report for trial 5:

	precision	recall	f1-score	support
0	0.97	1.00	0.98	5388
1	0.94	0.71	0.81	519
accuracy			0.97	5907
macro avg	0.96	0.85	0.90	5907
weighted avg	0.97	0.97	0.97	5907

accuracy_score for trial 5:
0.9707127137294735
ROC_AUC score :0.8542275491243655

In []:

In [16]:

```
##### Naive Bayes ##### block1 recall the required packages for later use
from sklearn.naive_bayes import GaussianNB
from math import sqrt
from math import pi
from math import exp
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
import warnings
import sklearn.exceptions
warnings.filterwarnings("ignore", category=sklearn.exceptions.UndefinedMetricWarning)
```

In [17]:

```
##### Naive Bayes ##### block2
# Cross validate model with Kfold cross val
kfold = KFold(n_splits=5, random_state=None, shuffle=False)
clf = GaussianNB()
parameters = {
    'var_smoothing': [1e-9,2e-9,2e-9]
}

grid_search = GridSearchCV(clf , param_grid=parameters, cv=kfold, verbose=2)
```

In [18]:

```
#####
# Naive Bayes #####
##### block3 run trials #####
##### use grid_search to train data #####
##### data1 trial1 #####
print('Trial 1')
grid_search.fit(data1_train_X[0],data1_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data1_test_X[0].astype('int')) # fit test set
nb_clf.score(data1_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data1_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data1_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data1_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data1_test_Y[0].astype('int'),y_pred_t1)))
#####
# Naive Bayes #####
##### block3 run trials #####
##### use grid_search to train data #####
##### data1 trial2 #####
print('Trial 2')
grid_search.fit(data1_train_X[1],data1_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data1_test_X[1].astype('int')) # fit test set
nb_clf.score(data1_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data1_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data1_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data1_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data1_test_Y[1].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data #####
##### data1 trial3 #####
print('Trial 3')
grid_search.fit(data1_train_X[2],data1_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data1_test_X[2].astype('int')) # fit test set
nb_clf.score(data1_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data1_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data1_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
```

```
print(accuracy_score(data1_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data1_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data ##### data1 trial4 #####
print('Trial 4')
grid_search.fit(data1_train_X[3],data1_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data1_test_X[3].astype('int')) # fit test set
nb_clf.score(data1_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data1_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data1_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data1_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data1_test_Y[3].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data ##### data1 trial5 #####
print('Trial 5')
grid_search.fit(data1_train_X[4],data1_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data1_test_X[4].astype('int')) # fit test set
nb_clf.score(data1_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data1_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data1_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data1_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data1_test_Y[4].astype('int'),y_pred_t1)))
```

```

Trial 1
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9801492537313432
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 1:
[[ 0 2113]
 [ 0 1187]]
classification_report for trial 1:
      precision    recall   f1-score   support
          0       0.00     0.00     0.00      2113
          1       0.36     1.00     0.53      1187
accuracy                           0.36      3300
macro avg       0.18     0.50     0.26      3300
weighted avg     0.13     0.36     0.19      3300

accuracy_score for trial 1:
0.3596969696969697
ROC_AUC score :0.5
Trial 2
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total

```

```

time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9786567164179104
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 2:
[[ 0 2113]
 [ 0 1187]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0       0.00     0.00     0.00      2113
          1       0.36     1.00     0.53      1187

accuracy                           0.36      3300
macro avg       0.18     0.50     0.26      3300
weighted avg    0.13     0.36     0.19      3300

accuracy_score for trial 2:
0.3596969696969697
ROC_AUC score :0.5
Trial 3
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9762686567164179
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 3:
[[ 0 2129
 [ 0 1171]]
classification_report for trial 3:
      precision    recall   f1-score   support
          0       0.00     0.00     0.00      2129
          1       0.35     1.00     0.52      1171
accuracy                           0.35      3300
macro avg       0.18     0.50     0.26      3300
weighted avg      0.13     0.35     0.19      3300
accuracy_score for trial 3:
0.35484848484848486
ROC_AUC score :0.5
Trial 4
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
```

```

time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9783582089552239
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 4:
[[ 0 2120]
 [ 0 1180]]
classification_report for trial 4:
      precision    recall   f1-score   support
          0         0.00     0.00     0.00      2120
          1         0.36     1.00     0.53      1180

accuracy                           0.36      3300
macro avg       0.18     0.50     0.26      3300
weighted avg    0.13     0.36     0.19      3300

accuracy_score for trial 4:
0.3575757575757576
ROC_AUC score :0.5
Trial 5
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9800000000000001
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 5:
[[ 0 2121]
 [ 0 1179]]
classification_report for trial 5:
      precision    recall   f1-score   support
          0         0.00     0.00     0.00     2121
          1         0.36     1.00     0.53     1179
accuracy                           0.36     3300
macro avg       0.18     0.50     0.26     3300
weighted avg    0.13     0.36     0.19     3300

accuracy_score for trial 5:
0.3572727272727273
ROC_AUC score :0.5
```

In [19]:

```
#####
# Naive Bayes #####
##### block4 run trials
# Cross validate model with Kfold cross val
kfold = KFold(n_splits=5, random_state= None, shuffle=False)
clf = GaussianNB()
parameters = {
    'var_smoothing': [1e-9,2e-9,2e-9]
}

grid_search = GridSearchCV(clf , param_grid=parameters, cv=kfold, verbose=2)
#####
# use grid_search to train data #####
data2 trial1 #####
print('Trial 1')
grid_search.fit(data2_train_X[0],data2_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data2_test_X[0].astype('int')) # fit test set
nb_clf.score(data2_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data2_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data2_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data2_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data2_test_Y[0].astype('int'),y_pred_t1)))
#####
# Naive Bayes #####
##### block3 run trials
#####
# use grid_search to train data #####
data2 trial2 #####
print('Trial 2')
grid_search.fit(data2_train_X[1],data2_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data2_test_X[1].astype('int')) # fit test set
nb_clf.score(data2_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data2_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data2_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data2_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data2_test_Y[1].astype('int'),y_pred_t1)))
#####
# use grid_search to train data #####
data2 trial3 #####
print('Trial 3')
grid_search.fit(data2_train_X[2],data2_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
```

```
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data2_test_X[2].astype('int')) # fit test set
nb_clf.score(data2_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data2_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data2_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
print(accuracy_score(data2_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data2_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data #####
print('Trial 4')
grid_search.fit(data2_train_X[3],data2_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data2_test_X[3].astype('int')) # fit test set
nb_clf.score(data2_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data2_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data2_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data2_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data2_test_Y[3].astype('int'),y_pred_t1)))
##### use grid_search to train data #####
print('Trial 5')
grid_search.fit(data2_train_X[4],data2_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data2_test_X[4].astype('int')) # fit test set
nb_clf.score(data2_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data2_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data2_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data2_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data2_test_Y[4].astype('int'),y_pred_t1)))
```

```

Trial 1
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9988059701492537
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 1:
[[3186 0]
 [ 1 113]]
classification_report for trial 1:
      precision    recall   f1-score   support
          0       1.00     1.00     1.00      3186
          1       1.00     0.99     1.00      114
accuracy                           1.00      3300
macro avg       1.00     1.00     1.00      3300
weighted avg     1.00     1.00     1.00      3300

accuracy_score for trial 1:
0.9996969696969698
ROC_AUC score :0.9956140350877193
Trial 2
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total

```

```

time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9974626865671642
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 2:
[[3178 7]
 [ 1 114]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0       1.00     1.00     1.00     3185
          1       0.94     0.99     0.97     115
accuracy                           1.00     3300
macro avg       0.97     0.99     0.98     3300
weighted avg     1.00     1.00     1.00     3300

accuracy_score for trial 2:
0.9975757575757576
ROC_AUC score :0.9945532728141424
Trial 3
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9973134328358209
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 3:
[[3181 5]
 [ 3 111]]
classification_report for trial 3:
      precision    recall   f1-score   support
          0       1.00     1.00     1.00     3186
          1       0.96     0.97     0.97      114
accuracy                           1.00     3300
macro avg       0.98     0.99     0.98     3300
weighted avg     1.00     1.00     1.00     3300

accuracy_score for trial 3:
0.9975757575757576
ROC_AUC score :0.9860574222750852
Trial 4
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
```

```

time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9991044776119402
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 4:
[[3194 0]
 [ 4 102]]
classification_report for trial 4:
      precision    recall   f1-score   support
          0       1.00     1.00     1.00      3194
          1       1.00     0.96     0.98      106
accuracy                           1.00      3300
macro avg       1.00     0.98     0.99      3300
weighted avg     1.00     1.00     1.00      3300

accuracy_score for trial 4:
0.9987878787878788
ROC_AUC score :0.9811320754716981
Trial 5
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9973134328358209
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 5:
[[3165 5]
 [ 3 127]]
classification_report for trial 5:
      precision    recall  f1-score   support

          0       1.00     1.00     1.00      3170
          1       0.96     0.98     0.97      130

   accuracy                           1.00      3300
  macro avg       0.98     0.99     0.98      3300
weighted avg       1.00     1.00     1.00      3300

accuracy_score for trial 5:
0.99757575757576
ROC_AUC score :0.9876728949284155
```

In [20]:

```
#####
# Naive Bayes #####
##### block5 run trials #####
##### use grid_search to train data #####
##### data3 trial1 #####
print('Trial 1')
grid_search.fit(data3_train_X[0],data3_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data3_test_X[0].astype('int')) # fit test set
nb_clf.score(data3_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data3_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data3_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data3_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data3_test_Y[0].astype('int'),y_pred_t1)))
#####
# Naive Bayes #####
##### block3 run trials #####
##### use grid_search to train data #####
##### data3 trial2 #####
print('Trial 2')
grid_search.fit(data3_train_X[1],data3_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data3_test_X[1].astype('int')) # fit test set
nb_clf.score(data3_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data3_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data3_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data3_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data3_test_Y[1].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data #####
##### data3 trial3 #####
print('Trial 3')
grid_search.fit(data3_train_X[2],data3_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data3_test_X[2].astype('int')) # fit test set
nb_clf.score(data3_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data3_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data3_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
```

```
print(accuracy_score(data3_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data3_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data ##### data3 trial4 #####
print('Trial 4')
grid_search.fit(data3_train_X[3],data3_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data3_test_X[3].astype('int')) # fit test set
nb_clf.score(data3_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data3_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data3_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data3_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data3_test_Y[3].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data ##### data3 trial5 #####
print('Trial 5')
grid_search.fit(data3_train_X[4],data3_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data3_test_X[4].astype('int')) # fit test set
nb_clf.score(data3_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data3_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data3_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data3_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data3_test_Y[4].astype('int'),y_pred_t1)))
```

```

Trial 1
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9715506788831314
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 1:
[[2579 99]
 [ 3 886]]
classification_report for trial 1:
      precision    recall   f1-score   support
          0       1.00     0.96     0.98     2678
          1       0.90     1.00     0.95     889

      accuracy                           0.97     3567
      macro avg       0.95     0.98     0.96     3567
      weighted avg    0.97     0.97     0.97     3567

accuracy_score for trial 1:
0.9714045416316233
ROC_AUC score :0.9798287676699112
Trial 2
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total

```

```

time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9679599952720299
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 2:
[[2600 104]
 [ 2 861]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0       1.00     0.96     0.98     2704
          1       0.89     1.00     0.94      863
accuracy                           0.97     3567
macro avg       0.95     0.98     0.96     3567
weighted avg    0.97     0.97     0.97     3567

accuracy_score for trial 2:
0.9702831511073732
ROC_AUC score :0.9796104822176664
Trial 3
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9704463737612908
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 3:
[[2557 96]
 [ 2 912]]
classification_report for trial 3:
      precision    recall   f1-score   support
          0       1.00     0.96     0.98     2653
          1       0.90     1.00     0.95     914
accuracy                           0.97     3567
macro avg       0.95     0.98     0.97     3567
weighted avg    0.98     0.97     0.97     3567

accuracy_score for trial 3:
0.9725259321558732
ROC_AUC score :0.9808131828795443
Trial 4
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
```

```

time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9700317231544713
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 4:
[[2597 98]
 [ 2 870]]
classification_report for trial 4:
      precision    recall   f1-score   support
          0       1.00     0.96     0.98     2695
          1       0.90     1.00     0.95     872

accuracy                           0.97    3567
macro avg       0.95     0.98     0.96    3567
weighted avg    0.97     0.97     0.97    3567

accuracy_score for trial 4:
0.9719652368937483
ROC_AUC score :0.9806713928273562
Trial 5
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9657498598766916
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 5:
[[2571  92]
 [  4 900]]
classification_report for trial 5:
      precision    recall  f1-score   support

          0       1.00     0.97     0.98     2663
          1       0.91     1.00     0.95     904

   accuracy                           0.97     3567
  macro avg       0.95     0.98     0.97     3567
weighted avg       0.98     0.97     0.97     3567

accuracy_score for trial 5:
0.9730866274179983
ROC_AUC score :0.9805138592112826
```

In [21]:

```
#####
# Naive Bayes #####
##### block6 run trials #####
##### use grid_search to train data #####
##### data4 trial1 #####
print('Trial 1')
grid_search.fit(data4_train_X[0],data4_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data4_test_X[0].astype('int')) # fit test set
nb_clf.score(data4_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data4_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data4_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data4_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data4_test_Y[0].astype('int'),y_pred_t1)))
#####
# Naive Bayes #####
##### block3 run trials #####
##### use grid_search to train data #####
##### data4 trial2 #####
print('Trial 2')
grid_search.fit(data4_train_X[1],data4_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data4_test_X[1].astype('int')) # fit test set
nb_clf.score(data4_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data4_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data4_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data4_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data4_test_Y[1].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data #####
##### data4 trial3 #####
print('Trial 3')
grid_search.fit(data4_train_X[2],data4_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data4_test_X[2].astype('int')) # fit test set
nb_clf.score(data4_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data4_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data4_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
```

```

print(accuracy_score(data4_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data4_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data ##### data4 trial4 #####
print('Trial 4')
grid_search.fit(data4_train_X[3],data4_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data4_test_X[3].astype('int')) # fit test set
nb_clf.score(data4_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data4_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data4_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data4_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data4_test_Y[3].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data ##### data4 trial5 #####
print('Trial 5')
grid_search.fit(data4_train_X[4],data4_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data4_test_X[4].astype('int')) # fit test set
nb_clf.score(data4_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data4_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data4_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data4_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data4_test_Y[4].astype('int'),y_pred_t1)))

```

```

Trial 1
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.671044776119403
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 1:
[[2171 0]
 [1129 0]]
classification_report for trial 1:
      precision    recall   f1-score   support
      -1       0.66     1.00     0.79     2171
       0       0.00     0.00     0.00     1129
accuracy                           0.66     3300
macro avg       0.33     0.50     0.40     3300
weighted avg     0.43     0.66     0.52     3300

accuracy_score for trial 1:
0.6578787878787878
ROC_AUC score :0.5
Trial 2
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total

```

```

time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.6659701492537314
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 2:
[[2205 0]
 [1095 0]]
classification_report for trial 2:
      precision    recall   f1-score   support
      -1       0.67     1.00     0.80     2205
       0       0.00     0.00     0.00     1095

accuracy                           0.67     3300
macro avg       0.33     0.50     0.40     3300
weighted avg     0.45     0.67     0.54     3300

accuracy_score for trial 2:
0.6681818181818182
ROC_AUC score :0.5
Trial 3
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.6761194029850746
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 3:
[[2137 0]
 [1163 0]]
classification_report for trial 3:
      precision    recall   f1-score   support
      -1       0.65     1.00     0.79     2137
       0       0.00     0.00     0.00     1163
accuracy                           0.65     3300
macro avg       0.32     0.50     0.39     3300
weighted avg     0.42     0.65     0.51     3300
accuracy_score for trial 3:
0.6475757575757576
ROC_AUC score :0.5
Trial 4
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
```

```

time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.6685074626865671
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 4:
[[2188 0]
 [1112 0]]
classification_report for trial 4:
      precision    recall   f1-score   support
          -1       0.66     1.00     0.80     2188
           0       0.00     0.00     0.00     1112

accuracy                           0.66     3300
macro avg       0.33     0.50     0.40     3300
weighted avg     0.44     0.66     0.53     3300

accuracy_score for trial 4:
0.6630303030303031
ROC_AUC score :0.5
Trial 5
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total  
time= 0.0s  
[CV] END .....var_smoothing=2e-09; total  
time= 0.0s  
best_estimator  
GaussianNB()  
best_score  
0.6737313432835821  
best_parameters  
{'var_smoothing': 1e-09}  
confusion_matrix for trial 5:  
[[2153 0]  
 [1147 0]]  
classification_report for trial 5:  
 precision recall f1-score support  
  
 -1 0.65 1.00 0.79 2153  
 0 0.00 0.00 0.00 1147  
  
 accuracy 0.65 3300  
 macro avg 0.33 0.50 0.39 3300  
 weighted avg 0.43 0.65 0.52 3300  
  
accuracy_score for trial 5:  
0.6524242424242425  
ROC_AUC score :0.5
```

In [22]:

```
#####
# Naive Bayes #####
##### block7 run trials #####
##### use grid_search to train data #####
##### data5 trial1 #####
print('Trial 1')
grid_search.fit(data5_train_X[0],data5_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data5_test_X[0].astype('float').astype('int')) # fit test set
nb_clf.score(data5_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data5_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data5_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data5_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data5_test_Y[0].astype('int'),y_pred_t1)))
#####
# Naive Bayes #####
##### block3 run trials #####
##### use grid_search to train data #####
##### data5 trial2 #####
print('Trial 2')
grid_search.fit(data5_train_X[1],data5_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data5_test_X[1].astype('float').astype('int')) # fit test set
nb_clf.score(data5_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data5_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data5_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data5_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data5_test_Y[1].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data #####
##### data5 trial3 #####
print('Trial 3')
grid_search.fit(data5_train_X[2],data5_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data5_test_X[2].astype('float').astype('int')) # fit test set
nb_clf.score(data5_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data5_test_Y[2].astype('int'),y_pred_t1))
```

```
print('classification_report for trial 3: ')
print(classification_report(data5_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3: ')
print(accuracy_score(data5_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data5_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data #####
print('Trial 4')
grid_search.fit(data5_train_X[3],data5_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data5_test_X[3].astype('float').astype('int')) # fit test set
nb_clf.score(data5_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data5_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4: ')
print(classification_report(data5_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4: ')
print(accuracy_score(data5_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data5_test_Y[3].astype('int'),y_pred_t1)))
#####
print('Trial 5')
grid_search.fit(data5_train_X[4],data5_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data5_test_X[4].astype('float').astype('int')) # fit test set
nb_clf.score(data5_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5: ')
print(confusion_matrix(data5_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5: ')
print(classification_report(data5_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5: ')
print(accuracy_score(data5_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data5_test_Y[4].astype('int'),y_pred_t1)))
```

Trial 1

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.4901365142320603
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 1:
[[ 717 2008]
 [ 490 1729]]
classification_report for trial 1:
      precision    recall   f1-score   support
          0       0.59      0.26      0.36      2725
          1       0.46      0.78      0.58      2219

      accuracy                           0.49      4944
     macro avg       0.53      0.52      0.47      4944
  weighted avg       0.54      0.49      0.46      4944

accuracy_score for trial 1:
0.4947411003236246
ROC_AUC score :0.521149538390299
Trial 2
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
```

```

time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.4554576909104985
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 2:
[[ 50 2656]
 [ 72 2166]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0       0.41      0.02      0.04     2706
          1       0.45      0.97      0.61     2238

accuracy                           0.45      4944
macro avg       0.43      0.49      0.32      4944
weighted avg    0.43      0.45      0.30      4944

accuracy_score for trial 2:
0.4482200647249191
ROC_AUC score :0.49315293786620534
Trial 3
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.4602432323521063
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 3:
[[ 174 2559]
 [ 99 2112]]
classification_report for trial 3:
      precision    recall   f1-score   support
          0       0.64     0.06     0.12      2733
          1       0.45     0.96     0.61      2211
accuracy                           0.46      4944
macro avg       0.54     0.51     0.36      4944
weighted avg    0.55     0.46     0.34      4944

accuracy_score for trial 3:
0.46237864077669905
ROC_AUC score :0.5094450906827007
Trial 4
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
```

```

time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.489839744162364
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 4:
[[ 142 2613]
 [ 91 2098]]
classification_report for trial 4:
      precision    recall   f1-score   support
          0       0.61     0.05     0.10     2755
          1       0.45     0.96     0.61     2189

accuracy                           0.45      4944
macro avg       0.53     0.50     0.35      4944
weighted avg    0.54     0.45     0.32      4944

accuracy_score for trial 4:
0.45307443365695793
ROC_AUC score :0.5049855779474837
Trial 5
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.4810549530825378
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 5:
[[ 53 2675]
 [ 64 2152]]
classification_report for trial 5:
      precision    recall   f1-score   support
          0         0.45     0.02     0.04     2728
          1         0.45     0.97     0.61     2216
accuracy                           0.45     4944
macro avg       0.45     0.50     0.32     4944
weighted avg    0.45     0.45     0.29     4944

accuracy_score for trial 5:
0.44599514563106796
ROC_AUC score :0.49527364303333793
```

In [23]:

```
#####
# Naive Bayes #####
##### block8 run trials #####
##### use grid_search to train data #####
##### data6 trial1 #####
print('Trial 1')
grid_search.fit(data6_train_X[0],data6_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data6_test_X[0].astype('int')) # fit test set
nb_clf.score(data6_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data6_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data6_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data6_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data6_test_Y[0].astype('int'),y_pred_t1)))
#####
# Naive Bayes #####
##### block3 run trials #####
##### use grid_search to train data #####
##### data6 trial2 #####
print('Trial 2')
grid_search.fit(data6_train_X[1],data6_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data6_test_X[1].astype('int')) # fit test set
nb_clf.score(data6_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data6_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data6_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data6_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data6_test_Y[1].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data #####
##### data6 trial3 #####
print('Trial 3')
grid_search.fit(data6_train_X[2],data6_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data6_test_X[2].astype('int')) # fit test set
nb_clf.score(data6_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data6_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data6_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
```

```
print(accuracy_score(data6_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data6_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data ##### data6 trial4 #####
print('Trial 4')
grid_search.fit(data6_train_X[3],data6_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data6_test_X[3].astype('int')) # fit test set
nb_clf.score(data6_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data6_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data6_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data6_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data6_test_Y[3].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data ##### data6 trial5 #####
print('Trial 5')
grid_search.fit(data6_train_X[4],data6_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data6_test_X[4].astype('int')) # fit test set
nb_clf.score(data6_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data6_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data6_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data6_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data6_test_Y[4].astype('int'),y_pred_t1)))
```

```

Trial 1
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.4076119402985075
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 1:
[[2173 5517]
 [ 295 1915]]
classification_report for trial 1:
      precision    recall   f1-score   support
          0       0.88     0.28      0.43      7690
          1       0.26     0.87      0.40     2210
accuracy                           0.41      9900
macro avg       0.57     0.57      0.41      9900
weighted avg     0.74     0.41      0.42      9900

accuracy_score for trial 1:
0.4129292929292929
ROC_AUC score :0.574545304767901
Trial 2
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total

```

```

time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.1s
[CV] END .....var_smoothing=1e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.1s
best_estimator
GaussianNB()
best_score
0.37641791044776124
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 2:
[[1824 5882]
 [ 242 1952]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0       0.88     0.24      0.37     7706
          1       0.25     0.89      0.39     2194
accuracy                           0.38     9900
macro avg       0.57     0.56      0.38     9900
weighted avg     0.74     0.38      0.38     9900

accuracy_score for trial 2:
0.3814141414141414
ROC_AUC score :0.5631989279683803
Trial 3
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.1s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.37293532338308455
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 3:
[[1690 6096]
 [ 232 1882]]
classification_report for trial 3:
      precision    recall   f1-score   support
          0       0.88     0.22      0.35      7786
          1       0.24     0.89      0.37      2114

accuracy                           0.36      9900
macro avg       0.56     0.55      0.36      9900
weighted avg    0.74     0.36      0.35      9900

accuracy_score for trial 3:
0.3608080808080808
ROC_AUC score :0.5536558473703256
Trial 4
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
```

```

time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.37910447761194027
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 4:
[[1777 5901]
 [ 254 1968]]
classification_report for trial 4:
      precision    recall   f1-score   support
          0       0.87     0.23      0.37     7678
          1       0.25     0.89      0.39     2222

accuracy                           0.38     9900
macro avg       0.56     0.56      0.38     9900
weighted avg    0.73     0.38      0.37     9900

accuracy_score for trial 4:
0.3782828282828283
ROC_AUC score :0.5585645240741839
Trial 5
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.1s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.1s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.3738805970149254
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 5:
[[1744 5922]
 [ 244 1990]]
classification_report for trial 5:
      precision    recall   f1-score   support
          0         0.88     0.23      0.36      7666
          1         0.25     0.89      0.39      2234
accuracy                           0.38      9900
macro avg       0.56     0.56      0.38      9900
weighted avg    0.74     0.38      0.37      9900

accuracy_score for trial 5:
0.37717171717171716
ROC_AUC score :0.5591384576433138
```

In [24]:

```
#####
# Naive Bayes #####
##### block9 run trials #####
##### use grid_search to train data #####
##### data7 trial1 #####
print('Trial 1')
grid_search.fit(data7_train_X[0],data7_train_Y[0].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data7_test_X[0].astype('int')) # fit test set
nb_clf.score(data7_test_X[0], y_pred_t1)
print('confusion_matrix for trial 1:')
print(confusion_matrix(data7_test_Y[0].astype('int'),y_pred_t1))
print('classification_report for trial 1:')
print(classification_report(data7_test_Y[0].astype('int'),y_pred_t1))
print('accuracy_score for trial 1:')
print(accuracy_score(data7_test_Y[0].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data7_test_Y[0].astype('int'),y_pred_t1)))
#####
# Naive Bayes #####
##### block3 run trials #####
##### use grid_search to train data #####
##### data7 trial2 #####
print('Trial 2')
grid_search.fit(data7_train_X[1],data7_train_Y[1].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data7_test_X[1].astype('int')) # fit test set
nb_clf.score(data7_test_X[1], y_pred_t1)
print('confusion_matrix for trial 2:')
print(confusion_matrix(data7_test_Y[1].astype('int'),y_pred_t1))
print('classification_report for trial 2:')
print(classification_report(data7_test_Y[1].astype('int'),y_pred_t1))
print('accuracy_score for trial 2:')
print(accuracy_score(data7_test_Y[1].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data7_test_Y[1].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data #####
##### data7 trial3 #####
print('Trial 3')
grid_search.fit(data7_train_X[2],data7_train_Y[2].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data7_test_X[2].astype('int')) # fit test set
nb_clf.score(data7_test_X[2], y_pred_t1)
print('confusion_matrix for trial 3:')
print(confusion_matrix(data7_test_Y[2].astype('int'),y_pred_t1))
print('classification_report for trial 3:')
print(classification_report(data7_test_Y[2].astype('int'),y_pred_t1))
print('accuracy_score for trial 3:')
```

```

print(accuracy_score(data7_test_Y[2].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data7_test_Y[2].astype('int'),y_pred_t1)))
##### use grid_search to train data ##### data7 trial4 #####
print('Trial 4')
grid_search.fit(data7_train_X[3],data7_train_Y[3].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data7_test_X[3].astype('int')) # fit test set
nb_clf.score(data7_test_X[3], y_pred_t1)
print('confusion_matrix for trial 4:')
print(confusion_matrix(data7_test_Y[3].astype('int'),y_pred_t1))
print('classification_report for trial 4:')
print(classification_report(data7_test_Y[3].astype('int'),y_pred_t1))
print('accuracy_score for trial 4:')
print(accuracy_score(data7_test_Y[3].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data7_test_Y[3].astype('int'),y_pred_t1)))
#####
##### use grid_search to train data ##### data7 trial5 #####
print('Trial 5')
grid_search.fit(data7_train_X[4],data7_train_Y[4].astype('int').ravel())
print('best_estimator')
print(grid_search.best_estimator_)
print('best_score')
print(grid_search.best_score_) # hyperparameter
print('best_parameters')
print(grid_search.best_params_) # hyperparameter search done
nb_clf = grid_search.best_estimator_
y_pred_t1 = nb_clf.predict(data7_test_X[4].astype('int')) # fit test set
nb_clf.score(data7_test_X[4], y_pred_t1)
print('confusion_matrix for trial 5:')
print(confusion_matrix(data7_test_Y[4].astype('int'),y_pred_t1))
print('classification_report for trial 5:')
print(classification_report(data7_test_Y[4].astype('int'),y_pred_t1))
print('accuracy_score for trial 5:')
print(accuracy_score(data7_test_Y[4].astype('int'),y_pred_t1))
print("ROC_AUC score :" + str(roc_auc_score(data7_test_Y[4].astype('int'),y_pred_t1)))

```

```

Trial 1
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.945208891249864
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 1:
[[5145 229]
 [ 88 445]]
classification_report for trial 1:
      precision    recall   f1-score   support
          0       0.98     0.96     0.97     5374
          1       0.66     0.83     0.74      533
accuracy                           0.95     5907
macro avg       0.82     0.90     0.85     5907
weighted avg     0.95     0.95     0.95     5907

accuracy_score for trial 1:
0.9463348569493821
ROC_AUC score :0.8961421157110429
Trial 2
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total

```

```

time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.946209586215552
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 2:
[[5133 231]
 [ 98 445]]
classification_report for trial 2:
      precision    recall   f1-score   support
          0       0.98     0.96     0.97     5364
          1       0.66     0.82     0.73      543
accuracy                           0.94     5907
macro avg       0.82     0.89     0.85     5907
weighted avg    0.95     0.94     0.95     5907

accuracy_score for trial 2:
0.9443033688843745
ROC_AUC score :0.8882281508398532
Trial 3
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9469601074398181
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 3:
[[5144 226]
 [ 103 434]]
classification_report for trial 3:
      precision    recall   f1-score   support
          0         0.98     0.96     0.97     5370
          1         0.66     0.81     0.73      537
accuracy                           0.94     5907
macro avg       0.82     0.88     0.85     5907
weighted avg    0.95     0.94     0.95     5907

accuracy_score for trial 3:
0.9443033688843745
ROC_AUC score :0.8830540037243948
Trial 4
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
```

```

time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s
best_estimator
GaussianNB()
best_score
0.9439592393411071
best_parameters
{'var_smoothing': 1e-09}
confusion_matrix for trial 4:
[[5148 223]
 [ 88 448]]
classification_report for trial 4:
      precision    recall   f1-score   support
          0         0.98     0.96     0.97     5371
          1         0.67     0.84     0.74      536

accuracy                           0.95      5907
macro avg       0.83     0.90     0.86      5907
weighted avg    0.95     0.95     0.95      5907

accuracy_score for trial 4:
0.947350600981886
ROC_AUC score :0.8971508126839272
Trial 5
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV] END .....var_smoothing=1e-09; total
time= 0.0s
[CV] END .....var_smoothing=2e-09; total
time= 0.0s

```

```
[CV] END .....var_smoothing=2e-09; total  
time= 0.0s  
[CV] END .....var_smoothing=2e-09; total  
time= 0.0s  
best_estimator  
GaussianNB()  
best_score  
0.945625905428346  
best_parameters  
{'var_smoothing': 1e-09}  
confusion_matrix for trial 5:  
[[5155 233]  
 [ 90 429]]  
classification_report for trial 5:  
 precision recall f1-score support  
  
 0 0.98 0.96 0.97 5388  
 1 0.65 0.83 0.73 519  
  
accuracy 0.95 5907  
macro avg 0.82 0.89 0.85 5907  
weighted avg 0.95 0.95 0.95 5907  
  
accuracy_score for trial 5:  
0.9453191129168783  
ROC_AUC score :0.8916726744510387
```

In []:

In [117]:

```
from numpy import arange
import numpy as np
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV, RepeatedKFold, KFold
from sklearn.linear_model import Lasso, LogisticRegression
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectFromModel
from sklearn import decomposition, datasets
from sklearn import tree
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier
```

In [91]:

```
d1 = pd.read_csv("Data_for_UCI_named.csv")
d2 = pd.read_csv("ai4i2020.csv")

d3_1 = pd.read_table("datatest.txt", sep = ",")
d3_3 = pd.read_table("datatest2.txt", sep = ",")
d3_2 = pd.read_table("datatraining.txt", sep = ",")
d3_temp = pd.concat([d3_1,d3_2])
d3 = pd.concat([d3_temp,d3_2])
d4 = pd.read_csv("log2.csv")

col_names = ["AF3", "F7", "F3", "FC5", "T7", "P7", "O1", "O2", "P8", "T8", "FC6", "F4", "F8", "AF4", "eyeDetection"]
dataset5 = pd.read_csv("EEG Eye State.csv", names = col_names)
d5 = dataset5.drop([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16])

d6 = pd.read_excel("default of credit card clients.xls", index_col = 0)
d6 = d6.drop("ID", axis=0)

d7 = pd.read_csv("HTRU_2.csv", header = None)
#v
##dataset5 = pd.read_csv("in-vehicle-coupon-recommendation.csv").values
##dataset6_part1 = pd.read_csv("datatraining.csv")
##dataset6_part2 = pd.read_csv("datatest.csv")
##dataset6 = data6_part1.append(data6_part2).values
```

In [81]:

```
dataset1 = pd.read_csv("Data_for_UCI_named.csv").values
dataset2 = pd.read_csv("ai4i2020.csv").values

dataset3_1 = pd.read_table("datatest.txt",sep = ",")
dataset3_3 = pd.read_table("datatest2.txt",sep = ",")
dataset3_2 = pd.read_table("datatraining.txt",sep = ",")
dataset3_temp = pd.concat([dataset3_1,dataset3_2])
dataset3 = pd.concat([dataset3_temp,dataset3_2]).values
dataset4 = pd.read_csv("log2.csv").values
```

In []:

In [83]:

In [84]:

In []:

In [113]:

In [88]:

In []:

In []:

In []:

In [92]:

```
x1 = pd.DataFrame(data1_X, columns = ['tau1', 'tau2', 'tau3', 'tau4', 'p2', 'p3',
, 'p4', 'g1', 'g2', 'g3', 'g4'])
for i in range(5):
    sel_ = SelectFromModel(Lasso(alpha = 0.03125))
    sel_.fit(data1_train_X[i], data1_train_Y[i])
    sel_.get_support()
    d1 = pd.read_csv("Data_for_UCI_named.csv")
    selected_feat = x1.columns[(sel_.get_support())]
    print('{} th iteration: {}'.format(i+1))
    print('total features: {}'.format((x1.shape[1])))
    print('selected features: {}'.format(len(selected_feat)))
    print('features with coefficients shrank to zero: {}'.format(np.sum(sel_.estimator_.coef_ == 0)))
```

```
1 th iteration:
total features: 11
selected features: 0
features with coefficients shrank to zero: 11
2 th iteration:
total features: 11
selected features: 0
features with coefficients shrank to zero: 11
3 th iteration:
total features: 11
selected features: 0
features with coefficients shrank to zero: 11
4 th iteration:
total features: 11
selected features: 0
features with coefficients shrank to zero: 11
5 th iteration:
total features: 11
selected features: 0
features with coefficients shrank to zero: 11
```

In [93]:

```
# To do list for all of us:
# correlation matrix among all variables(see relationship between variables)
# also plot the predicted variable against every other variables to see which ones seems most predictive
# fit pca on the training set
# Fisher's score
```

Dataset 1

In [128]:

d1

Out[128]:

	tau1	tau2	tau3	tau4	p1	p2	p3	p4	
0	2.959060	3.079885	8.381025	9.780754	3.763085	-0.782604	-1.257395	-1.723086	0.65
1	9.304097	4.902524	3.047541	1.369357	5.067812	-1.940058	-1.872742	-1.255012	0.41
2	8.971707	8.848428	3.046479	1.214518	3.405158	-1.207456	-1.277210	-0.920492	0.16
3	0.716415	7.669600	4.486641	2.340563	3.963791	-1.027473	-1.938944	-0.997374	0.44
4	3.134112	7.608772	4.943759	9.857573	3.525811	-1.125531	-1.845975	-0.554305	0.79
...
9995	2.930406	9.487627	2.376523	6.187797	3.343416	-0.658054	-1.449106	-1.236256	0.60
9996	3.392299	1.274827	2.954947	6.894759	4.349512	-1.663661	-0.952437	-1.733414	0.50
9997	2.364034	2.842030	8.776391	1.008906	4.299976	-1.380719	-0.943884	-1.975373	0.48
9998	9.631511	3.994398	2.757071	7.821347	2.514755	-0.966330	-0.649915	-0.898510	0.36
9999	6.530527	6.781790	4.349695	8.673138	3.492807	-1.390285	-1.532193	-0.570329	0.07

10000 rows × 14 columns

In [62]:

```
x = d1.iloc[:,0:-1].values
y = d1.iloc[:,13].values
d1["stabf"] = d1["stabf"].replace({"stable": "1", "unstable": "0"})
d1["stabf"] = d1["stabf"].replace({"stable": "1", "unstable": "0"})
```

In [63]:

```
#split into 5 trial
x1_train,x1_test,y1_train,y1_test = train_test_split(x,y,test_size = 0.33,random_state = 0,shuffle = True)
x1_train2,x1_test2,y1_train2,y1_test2 = train_test_split(x,y,test_size = 0.33,random_state = 1,shuffle = True)
x1_train3,x1_test3,y1_train3,y1_test3 = train_test_split(x,y,test_size = 0.33,random_state = 1,shuffle = True)
x1_train4,x1_test4,y1_train4,y1_test4 = train_test_split(x,y,test_size = 0.33,random_state = 1,shuffle = True)
x1_train5,x1_test5,y1_train5,y1_test5 = train_test_split(x,y,test_size = 0.33,random_state = 1,shuffle = True)
```

In [183]:

```
# tuning the model
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x1_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [2,4,6,8,10,12,15,20,25,30,35,40,45]
parameters = dict(pca__n_components=n_components,
                  dec_tree__criterion=criterion,
                  dec_tree__max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x1_train, y1_train)
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])
```

Best Criterion: entropy

Best max_depth: 45

Best Number Of Components: 13

```
DecisionTreeClassifier(criterion='entropy', max_depth=45)
```

In []:

In [189]:

```
# tuning the model decision tree -1
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x1_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x1_train, y1_train)
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])
```

Best Criterion: entropy
 Best max_depth: 45
 Best Number Of Components: 12

DecisionTreeClassifier(criterion='entropy', max_depth=45)

In [6]:

```
# tuning the model
#-2
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x1_train2.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x1_train2, y1_train2)
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])# tuning the mode
1
```

Best Criterion: entropy
 Best max_depth: 60
 Best Number Of Components: 13

DecisionTreeClassifier(criterion='entropy', max_depth=60)

In [8]:

```

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x1_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x1_train3, y1_train3)
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])# tuning the mode 1

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x1_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x1_train4, y1_train4)
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x1_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x1_train5, y1_train5)
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_')

```

```

components'])

print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

Best Criterion: entropy
Best max_depth: 45
Best Number Of Components: 12

DecisionTreeClassifier(criterion='entropy', max_depth=45)
Best Criterion: entropy
Best max_depth: 40
Best Number Of Components: 13

DecisionTreeClassifier(criterion='entropy', max_depth=40)
Best Criterion: entropy
Best max_depth: 40
Best Number Of Components: 13

DecisionTreeClassifier(criterion='entropy', max_depth=40)

```

In [119]:

```

#tuning random forest dataset
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
rfc=RandomForestClassifier(random_state=1)
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x1_train, y1_train)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x1_train2, y1_train2)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x1_train3, y1_train3)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x1_train4, y1_train4)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x1_train5, y1_train5)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

```

Out[119]:

```

{'criterion': 'gini',
 'max_depth': 4,
 'max_features': 'auto',
 'n_estimators': 200}

```

In [120]:

```
print(CV_rfc.best_score_)
```

0.9998507462686568

In []:

In [146]:

```
#train model random forest
from sklearn.ensemble import RandomForestClassifier

#Trial 1
classifier = RandomForestClassifier(n_estimators = 50)
classifier.fit(x1_train,y1_train)
y1_pred = classifier.predict(x1_test)
```

In [147]:

```
pd.unique(d1["stabf"])
```

Out[147]:

array(['unstable', 'stable'], dtype=object)

In [148]:

```
pd.unique(y1_pred)
```

Out[148]:

array(['unstable', 'stable'], dtype=object)

In [126]:

```
## Model - random forest
print("Trial 1")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 4,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x1_train,y1_train)
y1_pred = classifier.predict(x1_test)
print(confusion_matrix(y1_test,y1_pred))
print(classification_report(y1_test,y1_pred))
print(accuracy_score(y1_test,y1_pred))
print("ROC_AUC score:"+str(roc_auc_score(y1_test,y1_pred)))

print("Trial 2")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 4,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x1_train2,y1_train2)
y1_pred2 = classifier.predict(x1_test2)
print(confusion_matrix(y1_test2,y1_pred2))
print(classification_report(y1_test2,y1_pred2))
print(accuracy_score(y1_test2,y1_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y1_test2,y1_pred2)))

print("Trial 3")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 4,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x1_train3,y1_train3)
y1_pred3 = classifier.predict(x1_test3)
print(confusion_matrix(y1_test3,y1_pred3))
print(classification_report(y1_test3,y1_pred3))
print(accuracy_score(y1_test3,y1_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y1_test3,y1_pred3)))

print("Trial 4")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 4,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x1_train4,y1_train4)
y1_pred4 = classifier.predict(x1_test4)
print(confusion_matrix(y1_test4,y1_pred4))
print(classification_report(y1_test4,y1_pred4))
print(accuracy_score(y1_test4,y1_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y1_test4,y1_pred4)))

print("Trial 5")
classifier = RandomForestClassifier(criterion = 'gini',
    max_depth = 4,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x1_train5,y1_train5)
```

```
y1_pred5 = classifier.predict(x1_test5)
print(confusion_matrix(y1_test5,y1_pred5))
print(classification_report(y1_test5,y1_pred5))
print(accuracy_score(y1_test5,y1_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y1_test5,y1_pred5)))
```

Trial 1

```
[[2098    1]
 [ 0 1201]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2099
1	1.00	1.00	1.00	1201

accuracy			1.00	3300
----------	--	--	------	------

macro avg	1.00	1.00	1.00	3300
weighted avg	1.00	1.00	1.00	3300

0.9996969696969698

ROC_AUC score:0.9997617913292044

Trial 2

```
[[2112    1]
 [ 0 1187]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2113
1	1.00	1.00	1.00	1187

accuracy			1.00	3300
----------	--	--	------	------

macro avg	1.00	1.00	1.00	3300
weighted avg	1.00	1.00	1.00	3300

0.9996969696969698

ROC_AUC score:0.9997633696166588

Trial 3

```
[[2112    1]
 [ 0 1187]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2113
1	1.00	1.00	1.00	1187

accuracy			1.00	3300
----------	--	--	------	------

macro avg	1.00	1.00	1.00	3300
weighted avg	1.00	1.00	1.00	3300

0.9996969696969698

ROC_AUC score:0.9997633696166588

Trial 4

```
[[2112    1]
 [ 0 1187]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2113
1	1.00	1.00	1.00	1187

accuracy			1.00	3300
----------	--	--	------	------

macro avg	1.00	1.00	1.00	3300
weighted avg	1.00	1.00	1.00	3300

0.9996969696969698

ROC_AUC score:0.9997633696166588

Trial 5

```
[[2112    1]
 [ 0 1187]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2113
1	1.00	1.00	1.00	1187

0	1.00	1.00	1.00	2113
1	1.00	1.00	1.00	1187
accuracy			1.00	3300
macro avg	1.00	1.00	1.00	3300
weighted avg	1.00	1.00	1.00	3300

0.9996969696969698

ROC_AUC score: 0.9997633696166588

Decision Tree

In [114]:

```
d1[ "stabf" ] = d1[ "stabf" ].replace({ "stable": "1", "unstable": "0" })
d1[ "stabf" ] = d1[ "stabf" ].replace({ "stable": "1", "unstable": "0" })
```

In [65]:

```
# Trial one
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=45)
classifier = classifier.fit(x1_train,y1_train)
y1_pred = classifier.predict(x1_test)
print("Trial 1")
print("Accuracy Score:",accuracy_score(y1_test,y1_pred))
print(classification_report(y1_test,y1_pred))
print("ROC_AUC score:"+str(roc_auc_score(y1_test,y1_pred)))

# Trial two
#from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=60)
classifier = classifier.fit(x1_train2,y1_train2)
y1_pred2 = classifier.predict(x1_test2)
print("Trial 2")
print("Accuracy Score:",accuracy_score(y1_test2,y1_pred2))
print(classification_report(y1_test2,y1_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y1_test2,y1_pred2)))
# Trial 3
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=45)
classifier = classifier.fit(x1_train3,y1_train3)
y1_pred3 = classifier.predict(x1_test3)
print("Trial 3")
print("Accuracy Score:",accuracy_score(y1_test,y1_pred))
print(classification_report(y1_test3,y1_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y1_test3,y1_pred3)))
# Trial 4
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=40)
classifier = classifier.fit(x1_train4,y1_train4)
y1_pred4 = classifier.predict(x1_test4)
print("Accuracy Score:",accuracy_score(y1_test4,y1_pred4))
print(classification_report(y1_test4,y1_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y1_test4,y1_pred4)))
# Trial 5
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=40)
classifier = classifier.fit(x1_train5,y1_train5)
y1_pred5 = classifier.predict(x1_test5)
print("Accuracy Score:",accuracy_score(y1_test5,y1_pred5))
print(classification_report(y1_test5,y1_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y1_test5,y1_pred5)))
```

Trial 1**Accuracy Score:** 0.9996969696969698

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2099
1	1.00	1.00	1.00	1201
accuracy			1.00	3300
macro avg	1.00	1.00	1.00	3300
weighted avg	1.00	1.00	1.00	3300

ROC_AUC score: 0.9997617913292044**Trial 2****Accuracy Score:** 0.9996969696969698

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2113
1	1.00	1.00	1.00	1187
accuracy			1.00	3300
macro avg	1.00	1.00	1.00	3300
weighted avg	1.00	1.00	1.00	3300

ROC_AUC score: 0.9997633696166588**Trial 3****Accuracy Score:** 0.9996969696969698

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2113
1	1.00	1.00	1.00	1187
accuracy			1.00	3300
macro avg	1.00	1.00	1.00	3300
weighted avg	1.00	1.00	1.00	3300

ROC_AUC score: 0.9997633696166588**Accuracy Score:** 0.9996969696969698

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2113
1	1.00	1.00	1.00	1187
accuracy			1.00	3300
macro avg	1.00	1.00	1.00	3300
weighted avg	1.00	1.00	1.00	3300

ROC_AUC score: 0.9997633696166588**Accuracy Score:** 0.9996969696969698

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2113
1	1.00	1.00	1.00	1187
accuracy			1.00	3300
macro avg	1.00	1.00	1.00	3300
weighted avg	1.00	1.00	1.00	3300

ROC_AUC score: 0.9997633696166588

In []:

In []:

DataSet 2

In [20]:

d2.head(10)

Out[20]:

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	0
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	0
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	0
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	0
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	0
5	6	M14865	M	298.1	308.6	1425	41.9	11	0	0
6	7	L47186	L	298.1	308.6	1558	42.4	14	0	0
7	8	L47187	L	298.1	308.6	1527	40.2	16	0	0
8	9	M14868	M	298.3	308.7	1667	28.6	18	0	0
9	10	M14869	M	298.5	309.0	1741	28.0	21	0	0

In [43]:

d2["Type"].unique()

Out[43]:

array(['M', 'L', 'H'], dtype=object)

In [21]:

d2["Type"] = d2["Type"].replace({ "L": "1", "M": "2", "H": "3" })
d2["Type"].unique()

Out[21]:

array(['2', '1', '3'], dtype=object)

In [22]:

```
x_2 = d2.iloc[:,[0,2,3,4,5,6,7,9,10,11,12,13]].values  
y_2 = d2.iloc[:,8].values  
y_2
```

Out[22]:

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [23]:

```
x2_train,x2_test,y2_train,y2_test = train_test_split(x_2,y_2,test_size = 0.33,random_state = 0)  
x2_train2,x2_test2,y2_train2,y2_test2 = train_test_split(x_2,y_2,test_size = 0.33,random_state = 1)  
x2_train3,x2_test3,y2_train3,y2_test3 = train_test_split(x_2,y_2,test_size = 0.33,random_state = 2)  
x2_train4,x2_test4,y2_train4,y2_test4 = train_test_split(x_2,y_2,test_size = 0.33,random_state = 3)  
x2_train5,x2_test5,y2_train5,y2_test5 = train_test_split(x_2,y_2,test_size = 0.33,random_state = 4)
```

In [24]:

```
# tuning the model Decision Tree-1
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x2_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x2_train, y2_train)
print("Trial 1")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x2_train2.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x2_train2, y2_train2)
print("Trial 2")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x2_train3.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x2_train3, y2_train3)
print("Trial 3")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
```

```

n'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x2_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x2_train4, y2_train4)
print("Trial 4")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x2_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS.fit(x2_train5, y2_train5)
print("Trial 5")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

```

```
Trial 1
Best Criterion: gini
Best max_depth: 40
Best Number Of Components: 7

DecisionTreeClassifier(max_depth=40)
Trial 2
Best Criterion: gini
Best max_depth: 60
Best Number Of Components: 12

DecisionTreeClassifier(max_depth=60)
Trial 3
Best Criterion: gini
Best max_depth: 45
Best Number Of Components: 7

DecisionTreeClassifier(max_depth=45)
Trial 4
Best Criterion: entropy
Best max_depth: 45
Best Number Of Components: 9

DecisionTreeClassifier(criterion='entropy', max_depth=45)
Trial 5
Best Criterion: gini
Best max_depth: 40
Best Number Of Components: 5

DecisionTreeClassifier(max_depth=40)
```

In [121]:

```
#tuning random forest dataset2
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
rfc=RandomForestClassifier(random_state=1)
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x2_train, y2_train)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x2_train2, y2_train2)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x2_train3, y2_train3)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x2_train4, y2_train4)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x2_train5, y2_train5)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

{'criterion': 'gini', 'max_depth': 5, 'max_features': 'auto', 'n_estimators': 200}
0.999522388059701
```

Random Forest

In []:

In [132]:

```
## Model - random forest -dataset 2
print("Trial 1")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 5,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x2_train,y2_train)
y1_pred = classifier.predict(x2_test)
print(confusion_matrix(y2_test,y2_pred))
print(classification_report(y2_test,y2_pred))
print(accuracy_score(y2_test,y2_pred))
print("ROC_AUC score:"+str(roc_auc_score(y2_test,y2_pred)))

print("Trial 2")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 5,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x2_train2,y2_train2)
y1_pred2 = classifier.predict(x2_test2)
print(confusion_matrix(y2_test2,y2_pred2))
print(classification_report(y2_test2,y2_pred2))
print(accuracy_score(y2_test2,y2_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y2_test2,y2_pred2)))

print("Trial 3")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 5,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x2_train3,y2_train3)
y1_pred3 = classifier.predict(x2_test3)
print(confusion_matrix(y2_test3,y2_pred3))
print(classification_report(y2_test3,y2_pred3))
print(accuracy_score(y2_test3,y2_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y2_test3,y2_pred3)))

print("Trial 4")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 4,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x2_train4,y2_train4)
y1_pred4 = classifier.predict(x2_test4)
print(confusion_matrix(y2_test4,y2_pred4))
print(classification_report(y2_test4,y2_pred4))
print(accuracy_score(y2_test4,y2_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y2_test4,y2_pred4)))

print("Trial 5")
classifier = RandomForestClassifier(criterion = 'gini',
    max_depth = 4,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x2_train5,y2_train5)
```

```
y1_pred5 = classifier.predict(x2_test5)
print(confusion_matrix(y2_test5,y2_pred5))
print(classification_report(y2_test5,y2_pred5))
print(accuracy_score(y2_test,y2_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y2_test5,y2_pred5)))
```

Trial 1

```
[[3195    0]
 [  6   99]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	3195
1	1.00	0.94	0.97	105

accuracy			1.00	3300
----------	--	--	------	------

macro avg	1.00	0.97	0.98	3300
weighted avg	1.00	1.00	1.00	3300

0.9981818181818182

ROC_AUC score:0.9714285714285714

Trial 2

```
[[3176    10]
 [  1  113]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	3186
1	0.92	0.99	0.95	114

accuracy			1.00	3300
----------	--	--	------	------

macro avg	0.96	0.99	0.98	3300
weighted avg	1.00	1.00	1.00	3300

0.9966666666666667

ROC_AUC score:0.9940446691115736

Trial 3

```
[[3183     2]
 [  1  114]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	3185
1	0.98	0.99	0.99	115

accuracy			1.00	3300
----------	--	--	------	------

macro avg	0.99	1.00	0.99	3300
weighted avg	1.00	1.00	1.00	3300

0.9990909090909091

ROC_AUC score:0.9953382021705004

Trial 4

```
[[3183     3]
 [  3  111]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	3186
1	0.97	0.97	0.97	114

accuracy			1.00	3300
----------	--	--	------	------

macro avg	0.99	0.99	0.99	3300
weighted avg	1.00	1.00	1.00	3300

0.9981818181818182

ROC_AUC score:0.9863712954703142

Trial 5

```
[[3194     0]
 [  4  102]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	1.00	1.00	1.00	3194
1	1.00	0.96	0.98	106
accuracy			1.00	3300
macro avg	1.00	0.98	0.99	3300
weighted avg	1.00	1.00	1.00	3300

0.93787878787879

ROC_AUC score: 0.9811320754716981

Decision Tree

In [57]:

[redacted]

In [58]:

[redacted]

In [55]:

```
# Trial one -- dataset2
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='gini', max_depth=40)
classifier = classifier.fit(x2_train,y2_train)
y2_pred = classifier.predict(x2_test)
print("Trial 1")
print("Accuracy Score:",accuracy_score(y2_test,y2_pred))
print(classification_report(y2_test,y2_pred))
print("ROC_AUC score:"+str(roc_auc_score(y2_test,y2_pred)))

# Trial two
#from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='gini', max_depth=60)
classifier = classifier.fit(x2_train2,y2_train2)
y2_pred2 = classifier.predict(x2_test2)
print("Trial 2")
print("Accuracy Score:",accuracy_score(y2_test2,y2_pred2))
print(classification_report(y2_test2,y2_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y2_test2,y2_pred2)))
# Trial 3
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='gini', max_depth=45)
classifier = classifier.fit(x2_train3,y2_train3)
y2_pred3 = classifier.predict(x2_test3)
print("Trial 3")
print("Accuracy Score:",accuracy_score(y2_test3,y2_pred3))
print(classification_report(y2_test3,y2_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y2_test3,y2_pred3)))
# Trial 4
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='gini', max_depth=45)
classifier = classifier.fit(x2_train4,y2_train4)
y2_pred4 = classifier.predict(x2_test4)
print("Accuracy Score:",accuracy_score(y2_test4,y2_pred4))
print(classification_report(y2_test4,y2_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y2_test4,y2_pred4)))
# Trial 5
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='gini', max_depth=40)
classifier = classifier.fit(x2_train5,y2_train5)
y2_pred5 = classifier.predict(x2_test5)
print("Accuracy Score:",accuracy_score(y2_test5,y2_pred5))
print(classification_report(y2_test5,y2_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y2_test5,y2_pred5)))
```

Trial 1**Accuracy Score:** 0.9981818181818182

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3195
1	1.00	0.94	0.97	105
accuracy			1.00	3300
macro avg	1.00	0.97	0.98	3300
weighted avg	1.00	1.00	1.00	3300

ROC_AUC score: 0.9714285714285714**Trial 2****Accuracy Score:** 0.9966666666666667

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3186
1	0.92	0.99	0.95	114
accuracy			1.00	3300
macro avg	0.96	0.99	0.98	3300
weighted avg	1.00	1.00	1.00	3300

ROC_AUC score: 0.9940446691115736**Trial 3****Accuracy Score:** 0.9990909090909091

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3185
1	0.98	0.99	0.99	115
accuracy			1.00	3300
macro avg	0.99	1.00	0.99	3300
weighted avg	1.00	1.00	1.00	3300

ROC_AUC score: 0.9953382021705004**Accuracy Score:** 0.9981818181818182

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3186
1	0.97	0.97	0.97	114
accuracy			1.00	3300
macro avg	0.99	0.99	0.99	3300
weighted avg	1.00	1.00	1.00	3300

ROC_AUC score: 0.9863712954703142**Accuracy Score:** 0.9987878787878788

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3194
1	1.00	0.96	0.98	106
accuracy			1.00	3300
macro avg	1.00	0.98	0.99	3300
weighted avg	1.00	1.00	1.00	3300

ROC_AUC score: 0.9811320754716981

Dataset 3

In [64]:

```
d3.head(10)
```

Out[64]:

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
140	2015-02-02 14:19:00	23.7000	26.272	585.200000	749.200000	0.004764	1
141	2015-02-02 14:19:59	23.7180	26.290	578.400000	760.400000	0.004773	1
142	2015-02-02 14:21:00	23.7300	26.230	572.666667	769.666667	0.004765	1
143	2015-02-02 14:22:00	23.7225	26.125	493.750000	774.750000	0.004744	1
144	2015-02-02 14:23:00	23.7540	26.200	488.600000	779.000000	0.004767	1
145	2015-02-02 14:23:59	23.7600	26.260	568.666667	790.000000	0.004779	1
146	2015-02-02 14:25:00	23.7300	26.290	536.333333	798.000000	0.004776	1
147	2015-02-02 14:25:59	23.7540	26.290	509.000000	797.000000	0.004783	1
148	2015-02-02 14:26:59	23.7540	26.350	476.000000	803.200000	0.004794	1
149	2015-02-02 14:28:00	23.7360	26.390	510.000000	809.000000	0.004796	1

In [66]:

```
d3.shape
```

Out[66]:

```
(18951, 7)
```

In [93]:

```
#split date column from datetime to month, day, hour, minute column
d3['date'] = pd.to_datetime(d3['date'], format='%Y-%m-%d %H:%M:%S', errors='coerce')
d3['year'] = d3['date'].dt.year
d3['month'] = d3['date'].dt.month
d3['day'] = d3['date'].dt.day
d3['hour'] = d3['date'].dt.hour
d3['minute'] = d3['date'].dt.minute
d3.head(15)
```

Out[93]:

		date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy	year
140		2015-02-02 14:19:00	23.7000	26.272	585.200000	749.200000	0.004764	1	201
141		2015-02-02 14:19:59	23.7180	26.290	578.400000	760.400000	0.004773	1	201
142		2015-02-02 14:21:00	23.7300	26.230	572.666667	769.666667	0.004765	1	201
143		2015-02-02 14:22:00	23.7225	26.125	493.750000	774.750000	0.004744	1	201
144		2015-02-02 14:23:00	23.7540	26.200	488.600000	779.000000	0.004767	1	201
145		2015-02-02 14:23:59	23.7600	26.260	568.666667	790.000000	0.004779	1	201
146		2015-02-02 14:25:00	23.7300	26.290	536.333333	798.000000	0.004776	1	201
147		2015-02-02 14:25:59	23.7540	26.290	509.000000	797.000000	0.004783	1	201
148		2015-02-02 14:26:59	23.7540	26.350	476.000000	803.200000	0.004794	1	201
149		2015-02-02 14:28:00	23.7360	26.390	510.000000	809.000000	0.004796	1	201
150		2015-02-02 14:29:00	23.7450	26.445	481.500000	815.250000	0.004809	1	201
151		2015-02-02 14:30:00	23.7000	26.560	481.800000	824.000000	0.004817	1	201
152		2015-02-02 14:31:00	23.7000	26.600	475.250000	832.000000	0.004824	1	201
153		2015-02-02 14:31:59	23.7000	26.700	469.000000	845.333333	0.004842	1	201
154		2015-02-02 14:32:59	23.7000	26.774	464.000000	852.400000	0.004856	1	201

In [94]:

```
d3.shape
```

Out[94]:

```
(18951, 12)
```

In []:

In [30]:

```
x_3 = d3.iloc[:,1:6].values  
y_3 = d3.iloc[:,6].values  
y_3
```

Out[30]:

```
array([1, 1, 1, ..., 1, 1, 1])
```

In [33]:

```
x3_train,x3_test,y3_train,y3_test = train_test_split(x_3,y_3,test_size = 0.33,random_state = 0)  
x3_train2,x3_test2,y3_train2,y3_test2 = train_test_split(x_3,y_3,test_size = 0.33,random_state = 1)  
x3_train3,x3_test3,y3_train3,y3_test3 = train_test_split(x_3,y_3,test_size = 0.33,random_state = 2)  
x3_train4,x3_test4,y3_train4,y3_test4 = train_test_split(x_3,y_3,test_size = 0.33,random_state = 3)  
x3_train5,x3_test5,y3_train5,y3_test5 = train_test_split(x_3,y_3,test_size = 0.33,random_state = 4)
```

In [34]:

```
# tuning the model Decision Tree -1
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x3_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x3_train, y3_train)
print("Trial 1")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x3_train2.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x3_train2, y3_train2)
print("Trial 2")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x3_train3.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x3_train3, y3_train3)
print("Trial 3")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
```

```

n'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x3_train4.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x3_train4, y3_train4)
print("Trial 4")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x3_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS.fit(x3_train5, y3_train5)
print("Trial 5")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

```

```
Trial 1
Best Criterion: entropy
Best max_depth: 60
Best Number Of Components: 4

DecisionTreeClassifier(criterion='entropy', max_depth=60)
Trial 2
Best Criterion: entropy
Best max_depth: 60
Best Number Of Components: 4

DecisionTreeClassifier(criterion='entropy', max_depth=60)
Trial 3
Best Criterion: entropy
Best max_depth: 40
Best Number Of Components: 4

DecisionTreeClassifier(criterion='entropy', max_depth=40)
Trial 4
Best Criterion: entropy
Best max_depth: 60
Best Number Of Components: 5

DecisionTreeClassifier(criterion='entropy', max_depth=60)
Trial 5
Best Criterion: entropy
Best max_depth: 40
Best Number Of Components: 3

DecisionTreeClassifier(criterion='entropy', max_depth=40)
```

In [123]:

```
#tuning random forest dataset3
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
rfc=RandomForestClassifier(random_state=1)
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x3_train, y3_train)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x3_train2, y3_train2)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x3_train3, y3_train3)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x3_train4, y3_train4)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x3_train5, y3_train5)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

{'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 200}
0.9925967505341864
```

Random Forest

In [99]:

```
#train model random forest
from sklearn.ensemble import RandomForestClassifier

#Trial 1
classifier = RandomForestClassifier(n_estimators = 50)
classifier.fit(x3_train,y3_train)
y3_pred = classifier.predict(x3_test)
```

In [133]:

```
## Model - random forest -dataset 3
print("Trial 1")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x3_train,y3_train)
y3_pred = classifier.predict(x3_test)
print(confusion_matrix(y3_test,y3_pred))
print(classification_report(y3_test,y3_pred))
print(accuracy_score(y3_test,y3_pred))
print("ROC_AUC score:"+str(roc_auc_score(y3_test,y3_pred)))

print("Trial 2")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x3_train2,y3_train2)
y3_pred2 = classifier.predict(x3_test2)
print(confusion_matrix(y3_test2,y3_pred2))
print(classification_report(y3_test2,y3_pred2))
print(accuracy_score(y3_test2,y3_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y3_test2,y3_pred2)))

print("Trial 3")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x3_train3,y3_train3)
y3_pred3 = classifier.predict(x3_test3)
print(confusion_matrix(y3_test3,y3_pred3))
print(classification_report(y3_test3,y3_pred3))
print(accuracy_score(y3_test3,y3_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y3_test3,y3_pred3)))

print("Trial 4")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x3_train4,y3_train4)
y3_pred4 = classifier.predict(x3_test4)
print(confusion_matrix(y3_test4,y3_pred4))
print(classification_report(y3_test4,y3_pred4))
print(accuracy_score(y3_test4,y3_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y3_test4,y3_pred4)))

print("Trial 5")
classifier = RandomForestClassifier(criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x3_train5,y3_train5)
```

```
y3_pred5 = classifier.predict(x3_test5)
print(confusion_matrix(y3_test5,y3_pred5))
print(classification_report(y3_test5,y3_pred5))
print(accuracy_score(y3_test,y3_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y3_test5,y3_pred5)))
```

Trial 1

```
[[4802  27]
 [ 5 1420]]
```

	precision	recall	f1-score	support
0	1.00	0.99	1.00	4829
1	0.98	1.00	0.99	1425

accuracy		0.99	6254
----------	--	------	------

macro avg	0.99	1.00	0.99	6254
weighted avg	0.99	0.99	0.99	6254

0.9948832747041894

ROC_AUC score:0.9954500041779744

Trial 2

```
[[4758  35]
 [ 14 1447]]
```

	precision	recall	f1-score	support
0	1.00	0.99	0.99	4793
1	0.98	0.99	0.98	1461

accuracy		0.99	6254
----------	--	------	------

macro avg	0.99	0.99	0.99	6254
weighted avg	0.99	0.99	0.99	6254

0.9921650143907899

ROC_AUC score:0.9915576031838583

Trial 3

```
[[4761  44]
 [ 3 1446]]
```

	precision	recall	f1-score	support
0	1.00	0.99	1.00	4805
1	0.97	1.00	0.98	1449

accuracy		0.99	6254
----------	--	------	------

macro avg	0.98	0.99	0.99	6254
weighted avg	0.99	0.99	0.99	6254

0.9924848097217781

ROC_AUC score:0.9943862393167917

Trial 4

```
[[4718  28]
 [ 8 1500]]
```

	precision	recall	f1-score	support
0	1.00	0.99	1.00	4746
1	0.98	0.99	0.99	1508

accuracy		0.99	6254
----------	--	------	------

macro avg	0.99	0.99	0.99	6254
weighted avg	0.99	0.99	0.99	6254

0.994243684042213

ROC_AUC score:0.9943976275987263

Trial 5

```
[[4748  43]
 [ 14 1449]]
```

	precision	recall	f1-score	support
0				

0	1.00	0.99	0.99	4791
1	0.97	0.99	0.98	1463
accuracy			0.99	6254
macro avg	0.98	0.99	0.99	6254
weighted avg	0.99	0.99	0.99	6254

0.6461464662615926

ROC_AUC score: 0.9907277301239664

Decision Tree

In [104]:

In [58]:

```

# Trial one
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=60)
classifier = classifier.fit(x3_train,y3_train)
y3_pred = classifier.predict(x3_test)
print("Trial 1")
print("Accuracy Score:",accuracy_score(y3_test,y3_pred))
print(classification_report(y3_test,y3_pred))
print("ROC_AUC score:"+str(roc_auc_score(y3_test,y3_pred)))

# Trial two
#from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=60)
classifier = classifier.fit(x3_train2,y3_train2)
y3_pred2 = classifier.predict(x3_test2)
print("Trial 2")
print("Accuracy Score:",accuracy_score(y3_test2,y3_pred2))
print(classification_report(y3_test2,y3_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y3_test2,y3_pred2)))
# Trial 3
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=60)
classifier = classifier.fit(x3_train3,y3_train3)
y3_pred3 = classifier.predict(x3_test3)
print("Trial 3")
print("Accuracy Score:",accuracy_score(y3_test3,y3_pred3))
print(classification_report(y3_test3,y3_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y3_test3,y3_pred3)))
# Trial 4
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=40)
classifier = classifier.fit(x3_train4,y3_train4)
y3_pred4 = classifier.predict(x3_test4)
print("Accuracy Score:",accuracy_score(y3_test4,y3_pred4))
print(classification_report(y3_test4,y3_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y3_test4,y3_pred4)))
# Trial 5
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=60)
classifier = classifier.fit(x3_train5,y3_train5)
y3_pred5 = classifier.predict(x3_test5)
print("Accuracy Score:",accuracy_score(y3_test5,y3_pred5))
print(classification_report(y3_test5,y3_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y3_test5,y3_pred5)))

```

Trial 1**Accuracy Score:** 0.9974416373520947

	precision	recall	f1-score	support
0	1.00	1.00	1.00	4829
1	1.00	0.99	0.99	1425
accuracy			1.00	6254
macro avg	1.00	1.00	1.00	6254
weighted avg	1.00	1.00	1.00	6254

ROC_AUC score: 0.9958699814352613**Trial 2****Accuracy Score:** 0.9940837863767189

	precision	recall	f1-score	support
0	1.00	1.00	1.00	4793
1	0.99	0.98	0.99	1461
accuracy			0.99	6254
macro avg	0.99	0.99	0.99	6254
weighted avg	0.99	0.99	0.99	6254

ROC_AUC score: 0.990668215811531**Trial 3****Accuracy Score:** 0.996002558362648

	precision	recall	f1-score	support
0	1.00	1.00	1.00	4805
1	0.99	0.99	0.99	1449
accuracy			1.00	6254
macro avg	0.99	0.99	0.99	6254
weighted avg	1.00	1.00	1.00	6254

ROC_AUC score: 0.9945064557062927**Accuracy Score:** 0.9971218420211065

	precision	recall	f1-score	support
0	1.00	1.00	1.00	4746
1	1.00	0.99	0.99	1508
accuracy			1.00	6254
macro avg	1.00	1.00	1.00	6254
weighted avg	1.00	1.00	1.00	6254

ROC_AUC score: 0.9953891089075708**Accuracy Score:** 0.9936040933802367

	precision	recall	f1-score	support
0	0.99	1.00	1.00	4791
1	0.99	0.98	0.99	1463
accuracy			0.99	6254
macro avg	0.99	0.99	0.99	6254
weighted avg	0.99	0.99	0.99	6254

ROC_AUC score: 0.9901278784711537

Dataset 4

In [101]:

```
d4.head(10)
```

Out[101]:

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Action	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (se)
0	57222	53	54587	53	allow	177	94	83	2	119
1	56258	3389	56258	3389	allow	4768	1600	3168	19	-
2	6881	50321	43265	50321	allow	238	118	120	2	119
3	50553	3389	50553	3389	allow	3327	1438	1889	15	-
4	50002	443	45848	443	allow	25358	6778	18580	31	-
5	51465	443	39975	443	allow	3961	1595	2366	21	-
6	60513	47094	45469	47094	allow	320	140	180	6	-
7	50049	443	21285	443	allow	7912	3269	4643	23	119
8	52244	58774	2211	58774	allow	70	70	0	1	-
9	50627	443	16215	443	allow	8256	1674	6582	31	-

In [102]:

```
d4.shape
```

Out[102]:

```
(65532, 12)
```

In [93]:

```
d4[ "Action" ].unique()
```

Out[93]:

```
array(['allow', 'drop', 'deny', 'reset-both'], dtype=object)
```

In [92]:

```
#d4[ "Action" ] = d4[ "Action" ].replace({ "allow": "1", "deny": "0", "drop": "2", "reset-both": "3" }).astype( "int" )
x_4 = d4.iloc[:, [0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11] ].values
y_4 = d4.iloc[:, 4 ].values
y_4
```

Out[92]:

```
array(['allow', 'allow', 'allow', ..., 'drop', 'drop', 'drop'],
      dtype=object)
```

In [94]:

```
x4_train,x4_test,y4_train,y4_test = train_test_split(x_4,y_4,test_size = 0.33,random_state = 0)
x4_train2,x4_test2,y4_train2,y4_test2 = train_test_split(x_4,y_4,test_size = 0.33,random_state = 1)
x4_train3,x4_test3,y4_train3,y4_test3 = train_test_split(x_4,y_4,test_size = 0.33,random_state = 2)
x4_train4,x4_test4,y4_train4,y4_test4 = train_test_split(x_4,y_4,test_size = 0.33,random_state = 3)
x4_train5,x4_test5,y4_train5,y4_test5 = train_test_split(x_4,y_4,test_size = 0.33,random_state = 4)
```

In [37]:

```
# tuning the model Decision Tree -1
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x4_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x4_train, y4_train)
print("Trial 1")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

#Trial 2
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x4_train2.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x4_train2, y4_train2)
print("Trial 2")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x4_train3.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x4_train3, y4_train3)
print("Trial 3")
```

```

print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x4_train4.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x4_train4, y4_train4)
print("Trial 4")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x4_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS.fit(x4_train5, y4_train5)
print("Trial 5")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

```

```
Trial 1
Best Criterion: entropy
Best max_depth: 60
Best Number Of Components: 10

DecisionTreeClassifier(criterion='entropy', max_depth=60)
Trial 2
Best Criterion: entropy
Best max_depth: 60
Best Number Of Components: 11

DecisionTreeClassifier(criterion='entropy', max_depth=60)
Trial 3
Best Criterion: entropy
Best max_depth: 45
Best Number Of Components: 10

DecisionTreeClassifier(criterion='entropy', max_depth=45)
Trial 4
Best Criterion: entropy
Best max_depth: 60
Best Number Of Components: 9

DecisionTreeClassifier(criterion='entropy', max_depth=60)
Trial 5
Best Criterion: entropy
Best max_depth: 40
Best Number Of Components: 9

DecisionTreeClassifier(criterion='entropy', max_depth=40)
```

In [127]:

```
#tuning random forest dataset4
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
rfc=RandomForestClassifier(random_state=1)
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x4_train, y4_train)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x4_train2, y4_train2)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x_train3, y4_train3)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x4_train4, y4_train4)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x4_train5, y4_train5)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

{'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'n_estimators': 200}
0.9986334389862837
```

Random Forest

In [115]:

In [136]:

```
## Model - random forest -dataset 4
print("Trial 1")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 4,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x4_train,y4_train)
y4_pred = classifier.predict(x4_test)
print(confusion_matrix(y4_test,y4_pred))
print(classification_report(y4_test,y4_pred))
print(accuracy_score(y4_test,y4_pred))
#print("ROC_AUC score:"+str(roc_auc_score(y4_test,y4_pred)))

print("Trial 2")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x4_train2,y4_train2)
y4_pred2 = classifier.predict(x4_test2)
print(confusion_matrix(y4_test2,y4_pred2))
print(classification_report(y4_test2,y4_pred2))
print(accuracy_score(y4_test2,y4_pred2))
#print("ROC_AUC score:"+str(roc_auc_score(y4_test2,y4_pred2)))

print("Trial 3")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 5,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x4_train3,y4_train3)
y4_pred3 = classifier.predict(x4_test3)
print(confusion_matrix(y4_test3,y4_pred3))
print(classification_report(y4_test3,y4_pred3))
print(accuracy_score(y4_test3,y4_pred3))
#print("ROC_AUC score:"+str(roc_auc_score(y4_test3,y4_pred3)))

print("Trial 4")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x4_train4,y4_train4)
y4_pred4 = classifier.predict(x4_test4)
print(confusion_matrix(y4_test4,y4_pred4))
print(classification_report(y4_test4,y4_pred4))
print(accuracy_score(y4_test4,y4_pred4))
#print("ROC_AUC score:"+str(roc_auc_score(y4_test4,y4_pred4)))

print("Trial 5")
classifier = RandomForestClassifier(criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 200)
classifier.fit(x4_train5,y4_train5)
```

```
y4_pred5 = classifier.predict(x4_test5)
print(confusion_matrix(y4_test5,y4_pred5))
print(classification_report(y4_test5,y4_pred5))
print(accuracy_score(y4_test5,y4_pred5))
#print("ROC_AUC score:"+str(roc_auc_score(y4_test5,y4_pred5)))
```

Trial 1

```
[[12443      5      0      0]
 [    7  4894     28      0]
 [    0      0  4232      0]
 [    0     14      0      3]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

allow	1.00	1.00	1.00	12448
deny	1.00	0.99	0.99	4929
drop	0.99	1.00	1.00	4232
reset-both	1.00	0.18	0.30	17
accuracy			1.00	21626
macro avg	1.00	0.79	0.82	21626
weighted avg	1.00	1.00	1.00	21626

0.9975030056413576

Trial 2

```
[[12349      6      0      0]
 [    1  4976     9      0]
 [    0      0  4271      0]
 [    0     10      0      4]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

allow	1.00	1.00	1.00	12355
deny	1.00	1.00	1.00	4986
drop	1.00	1.00	1.00	4271
reset-both	1.00	0.29	0.44	14
accuracy			1.00	21626
macro avg	1.00	0.82	0.86	21626
weighted avg	1.00	1.00	1.00	21626

0.9987977434569499

Trial 3

```
[[12532      2      0      0]
 [    1  4865     21      0]
 [    0      0  4192      0]
 [    0     10      0      3]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

allow	1.00	1.00	1.00	12534
deny	1.00	1.00	1.00	4887
drop	1.00	1.00	1.00	4192
reset-both	1.00	0.23	0.38	13
accuracy			1.00	21626
macro avg	1.00	0.81	0.84	21626
weighted avg	1.00	1.00	1.00	21626

0.9984278183667807

Trial 4

```
[[12491      2      0      0]
 [    0  4844      8      0]
 [    0      0  4264      0]
 [    0     13      0      4]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

allow	1.00	1.00	1.00	12493
deny	1.00	1.00	1.00	4852
drop	1.00	1.00	1.00	4264

reset-both	1.00	0.24	0.38	17
accuracy			1.00	21626
macro avg	1.00	0.81	0.84	21626
weighted avg	1.00	1.00	1.00	21626

0.9989364653657634

Trial 5

```
[[12469      3      0      0]
 [  1   4917     12      0]
 [  0     0   4206      0]
 [  0    15      0     3]]
```

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	12472
deny	1.00	1.00	1.00	4930
drop	1.00	1.00	1.00	4206
reset-both	1.00	0.17	0.29	18
accuracy			1.00	21626
macro avg	1.00	0.79	0.82	21626
weighted avg	1.00	1.00	1.00	21626

0.9985665402755942

Decision Tree

In [118]:

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier = classifier.fit(x4_train,y4_train)
y4_pred = classifier.predict(x4_test)
```

In [112]:

```

# Trial one
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=60)
classifier = classifier.fit(x4_train,y4_train)
y4_pred = classifier.predict(x4_test)
print("Trial 1")
print("Accuracy Score:",accuracy_score(y4_test,y4_pred))
print(classification_report(y4_test,y4_pred))
#ras = roc_auc_score(y4_test,y4_pred)
#print("ROC_AUC score:"+str(roc_auc_score(y4_test2,y4_pred2,multi_class = 'ovr',
labels = ["deny", "drop", "allow", "reset-both"])))

# Trial two
#from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=60)
classifier = classifier.fit(x4_train2,y4_train2)
y4_pred2 = classifier.predict(x4_test2)
print("Trial 2")
print("Accuracy Score:",accuracy_score(y4_test2,y4_pred2))
print(classification_report(y4_test2,y4_pred2))
#print("ROC_AUC score:"+str(roc_auc_score(y4_test2,y4_pred2)))
# Trial 3
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=45)
classifier = classifier.fit(x4_train3,y4_train3)
y4_pred3 = classifier.predict(x4_test3)
print("Trial 3")
print("Accuracy Score:",accuracy_score(y4_test3,y4_pred3))
print(classification_report(y4_test3,y4_pred3))
#print("ROC_AUC score:"+str(roc_auc_score(y4_test3,y4_pred3)))
# Trial 4
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=60)
classifier = classifier.fit(x4_train4,y4_train4)
y4_pred4 = classifier.predict(x4_test4)
print("Accuracy Score:",accuracy_score(y4_test4,y4_pred4))
print(classification_report(y4_test4,y4_pred4))
#print("ROC_AUC score:"+str(roc_auc_score(y4_test4,y4_pred4)))
# Trial 5
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=45)
classifier = classifier.fit(x4_train5,y4_train5)
y4_pred5 = classifier.predict(x4_test5)
print("Accuracy Score:",accuracy_score(y4_test5,y4_pred5))
print(classification_report(y4_test5,y4_pred5))
#print("ROC_AUC score:"+str(roc_auc_score(y4_test5,y4_pred5)))

```

Trial 1**Accuracy Score:** 0.9977342088227134

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	12448
deny	1.00	1.00	1.00	4929
drop	1.00	1.00	1.00	4232
reset-both	0.46	0.35	0.40	17
accuracy			1.00	21626
macro avg	0.86	0.84	0.85	21626
weighted avg	1.00	1.00	1.00	21626

Trial 2**Accuracy Score:** 0.9980116526403403

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	12355
deny	1.00	1.00	1.00	4986
drop	1.00	1.00	1.00	4271
reset-both	0.50	0.50	0.50	14
accuracy			1.00	21626
macro avg	0.87	0.87	0.87	21626
weighted avg	1.00	1.00	1.00	21626

Trial 3**Accuracy Score:** 0.9977342088227134

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	12534
deny	0.99	1.00	0.99	4887
drop	1.00	1.00	1.00	4192
reset-both	0.45	0.38	0.42	13
accuracy			1.00	21626
macro avg	0.86	0.84	0.85	21626
weighted avg	1.00	1.00	1.00	21626

Accuracy Score: 0.9984278183667807

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	12493
deny	1.00	1.00	1.00	4852
drop	1.00	1.00	1.00	4264
reset-both	0.73	0.47	0.57	17
accuracy			1.00	21626
macro avg	0.93	0.87	0.89	21626
weighted avg	1.00	1.00	1.00	21626

Accuracy Score: 0.9977342088227134

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	12472
deny	0.99	1.00	1.00	4930
drop	1.00	1.00	1.00	4206
reset-both	0.64	0.50	0.56	18
accuracy			1.00	21626
macro avg	0.91	0.87	0.89	21626

weighted avg	1.00	1.00	1.00	21626
--------------	------	------	------	-------

Dataset 5

In [120]:

```
d5.head(5)
```

Out[120]:

	AF3	F7	F3	FC5	T7	P7	O1	O2	P8	T8
17	4329.23	4009.23	4289.23	4148.21	4350.26	4586.15	4096.92	4641.03	4222.05	4238.46
18	4324.62	4004.62	4293.85	4148.72	4342.05	4586.67	4097.44	4638.97	4210.77	4226.67
19	4327.69	4006.67	4295.38	4156.41	4336.92	4583.59	4096.92	4630.26	4207.69	4222.05
20	4328.72	4011.79	4296.41	4155.90	4343.59	4582.56	4097.44	4630.77	4217.44	4235.38
21	4326.15	4011.79	4292.31	4151.28	4347.69	4586.67	4095.90	4627.69	4210.77	4244.10

In [123]:

```
d5.shape
```

Out[123]:

```
(14980, 15)
```

In [38]:

```
x_5 = d5.iloc[:,0:14].values
y_5 = d5.iloc[:,14].values
y_5
```

Out[38]:

```
array([0., 0., 0., ..., 1., 1., 1.])
```

In [39]:

```
x5_train,x5_test,y5_train,y5_test = train_test_split(x_5,y_5,test_size = 0.33,random_state = 0)
x5_train2,x5_test2,y5_train2,y5_test2 = train_test_split(x_5,y_5,test_size = 0.33,random_state = 1)
x5_train3,x5_test3,y5_train3,y5_test3 = train_test_split(x_5,y_5,test_size = 0.33,random_state = 2)
x5_train4,x5_test4,y5_train4,y5_test4 = train_test_split(x_5,y_5,test_size = 0.33,random_state = 3)
x5_train5,x5_test5,y5_train5,y5_test5 = train_test_split(x_5,y_5,test_size = 0.33,random_state = 4)
```

In [40]:

```
# tuning the model Decision Tree -1
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x5_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x5_train, y5_train)
print("Trial 1")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

#Trial 2
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x5_train2.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60,70]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x5_train2, y5_train2)
print("Trial 2")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x5_train3.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60,70]
parameters = dict(pca__n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x5_train3, y5_train3)
print("Trial 3")
```

```

print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x4_train5.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60,70]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x5_train4, y5_train4)
print("Trial 4")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x5_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60,70]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS.fit(x5_train5, y5_train5)
print("Trial 5")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

```

```
Trial 1
Best Criterion: entropy
Best max_depth: 45
Best Number Of Components: 13

DecisionTreeClassifier(criterion='entropy', max_depth=45)
Trial 2
Best Criterion: entropy
Best max_depth: 40
Best Number Of Components: 13

DecisionTreeClassifier(criterion='entropy', max_depth=40)
Trial 3
Best Criterion: gini
Best max_depth: 40
Best Number Of Components: 14

DecisionTreeClassifier(max_depth=40)
Trial 4
Best Criterion: entropy
Best max_depth: 70
Best Number Of Components: 11

DecisionTreeClassifier(criterion='entropy', max_depth=70)
Trial 5
Best Criterion: entropy
Best max_depth: 60
Best Number Of Components: 11

DecisionTreeClassifier(criterion='entropy', max_depth=60)
```

In [128]:

```
#tuning random forest dataset5
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
rfc=RandomForestClassifier(random_state=43)
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x5_train, y5_train)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x5_train2, y5_train2)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x5_train3, y5_train3)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x5_train4, y5_train4)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x2_train5, y2_train5)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

{'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 500}
0.8291163695988345
```

In [138]:

```
#train model random forest
#from sklearn.ensemble import RandomForestClassifier

#Trial 1
## Model - random forest -dataset 5
print("Trial 1")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 4,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x5_train,y5_train)
y5_pred = classifier.predict(x5_test)
print(confusion_matrix(y5_test,y5_pred))
print(classification_report(y5_test,y5_pred))
print(accuracy_score(y5_test,y5_pred))
print("ROC_AUC score:"+str(roc_auc_score(y5_test,y5_pred)))

print("Trial 2")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x5_train2,y5_train2)
y5_pred2 = classifier.predict(x5_test2)
print(confusion_matrix(y5_test2,y5_pred2))
print(classification_report(y5_test2,y5_pred2))
print(accuracy_score(y5_test2,y5_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y5_test2,y5_pred2)))

print("Trial 3")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 5,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x5_train3,y5_train3)
y5_pred3 = classifier.predict(x5_test3)
print(confusion_matrix(y5_test3,y5_pred3))
print(classification_report(y5_test3,y5_pred3))
print(accuracy_score(y5_test3,y5_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y5_test3,y5_pred3)))

print("Trial 4")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x5_train4,y5_train4)
y5_pred4 = classifier.predict(x5_test4)
print(confusion_matrix(y5_test4,y5_pred4))
print(classification_report(y5_test4,y5_pred4))
print(accuracy_score(y5_test4,y5_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y5_test4,y5_pred4)))

print("Trial 5")
classifier = RandomForestClassifier(criterion = 'gini',
```

```
max_depth = 8,  
max_features = 'auto',  
n_estimators = 500)  
classifier.fit(x5_train5,y5_train5)  
y5_pred5 = classifier.predict(x5_test5)  
print(confusion_matrix(y5_test5,y5_pred5))  
print(classification_report(y5_test5,y5_pred5))  
print(accuracy_score(y5_test5,y5_pred5))  
print("ROC_AUC score:"+str(roc_auc_score(y5_test5,y5_pred5)))
```

Trial 1

```
[[2348  351]
 [1000 1245]]
```

	precision	recall	f1-score	support
0.0	0.70	0.87	0.78	2699
1.0	0.78	0.55	0.65	2245

accuracy			0.73	4944
----------	--	--	------	------

macro avg	0.74	0.71	0.71	4944
weighted avg	0.74	0.73	0.72	4944

0.7267394822006472

ROC_AUC score:0.7122587677858087

Trial 2

```
[[2523  202]
 [ 621 1598]]
```

	precision	recall	f1-score	support
0.0	0.80	0.93	0.86	2725
1.0	0.89	0.72	0.80	2219

accuracy			0.83	4944
----------	--	--	------	------

macro avg	0.85	0.82	0.83	4944
weighted avg	0.84	0.83	0.83	4944

0.8335355987055016

ROC_AUC score:0.8230078843681135

Trial 3

```
[[2323  383]
 [ 827 1411]]
```

	precision	recall	f1-score	support
0.0	0.74	0.86	0.79	2706
1.0	0.79	0.63	0.70	2238

accuracy			0.76	4944
----------	--	--	------	------

macro avg	0.76	0.74	0.75	4944
weighted avg	0.76	0.76	0.75	4944

0.7552588996763754

ROC_AUC score:0.7444681563559481

Trial 4

```
[[2523  210]
 [ 643 1568]]
```

	precision	recall	f1-score	support
0.0	0.80	0.92	0.86	2733
1.0	0.88	0.71	0.79	2211

accuracy			0.83	4944
----------	--	--	------	------

macro avg	0.84	0.82	0.82	4944
weighted avg	0.83	0.83	0.82	4944

0.827467637540453

ROC_AUC score:0.8161713635196931

Trial 5

```
[[2529  226]
 [ 577 1612]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0.0	0.81	0.92	0.86	2755
1.0	0.88	0.74	0.80	2189
accuracy			0.84	4944
macro avg	0.85	0.83	0.83	4944
weighted avg	0.84	0.84	0.84	4944

0.8375809061488673

ROC_AUC score: 0.8271883257236522

In [97]:

```

#Decision Tree
#from sklearn.tree import DecisionTreeClassifier
# Trial one
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=45)
classifier = classifier.fit(x5_train,y5_train)
y5_pred = classifier.predict(x5_test)
print("Trial 1")
print("Accuracy Score:",accuracy_score(y5_test,y5_pred))
print(classification_report(y5_test,y5_pred))
print("ROC_AUC score:"+str(roc_auc_score(y5_test,y5_pred)))

# Trial two
#from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=40)
classifier = classifier.fit(x5_train2,y5_train2)
y5_pred2 = classifier.predict(x5_test2)
print("Trial 2")
print("Accuracy Score:",accuracy_score(y5_test2,y5_pred2))
print(classification_report(y5_test2,y5_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y5_test2,y5_pred2)))
# Trial 3
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='gini', max_depth=40)
classifier = classifier.fit(x5_train3,y5_train3)
y5_pred3 = classifier.predict(x5_test3)
print("Trial 3")
print("Accuracy Score:",accuracy_score(y5_test3,y5_pred3))
print(classification_report(y5_test3,y5_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y5_test3,y5_pred3)))
# Trial 4
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=70)
classifier = classifier.fit(x5_train4,y5_train4)
y5_pred4 = classifier.predict(x5_test4)
print("Accuracy Score:",accuracy_score(y5_test4,y5_pred4))
print(classification_report(y5_test4,y5_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y5_test4,y5_pred4)))
# Trial 5
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=60)
classifier = classifier.fit(x5_train5,y5_train5)
y5_pred5 = classifier.predict(x5_test5)
print("Accuracy Score:",accuracy_score(y5_test5,y5_pred5))
print(classification_report(y5_test5,y5_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y5_test5,y5_pred5)))

```

Trial 1**Accuracy Score:** 0.837378640776699

	precision	recall	f1-score	support
0.0	0.85	0.85	0.85	2699
1.0	0.82	0.82	0.82	2245
accuracy			0.84	4944
macro avg	0.84	0.84	0.84	4944
weighted avg	0.84	0.84	0.84	4944

ROC_AUC score: 0.8359207526337809**Trial 2****Accuracy Score:** 0.8339401294498382

	precision	recall	f1-score	support
0.0	0.85	0.85	0.85	2725
1.0	0.82	0.81	0.81	2219
accuracy			0.83	4944
macro avg	0.83	0.83	0.83	4944
weighted avg	0.83	0.83	0.83	4944

ROC_AUC score: 0.8316592729182085**Trial 3****Accuracy Score:** 0.8379854368932039

	precision	recall	f1-score	support
0.0	0.85	0.86	0.85	2706
1.0	0.82	0.82	0.82	2238
accuracy			0.84	4944
macro avg	0.84	0.84	0.84	4944
weighted avg	0.84	0.84	0.84	4944

ROC_AUC score: 0.8359989418807178**Accuracy Score:** 0.8216019417475728

	precision	recall	f1-score	support
0.0	0.84	0.84	0.84	2733
1.0	0.80	0.80	0.80	2211
accuracy			0.82	4944
macro avg	0.82	0.82	0.82	4944
weighted avg	0.82	0.82	0.82	4944

ROC_AUC score: 0.8193748352340682**Accuracy Score:** 0.8230177993527508

	precision	recall	f1-score	support
0.0	0.85	0.83	0.84	2755
1.0	0.79	0.81	0.80	2189
accuracy			0.82	4944
macro avg	0.82	0.82	0.82	4944
weighted avg	0.82	0.82	0.82	4944

ROC_AUC score: 0.8219109903584909

Dataset 6

In [133]:

```
d6.head(10)
```

Out[133]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X15	X16	X17	X18	X19
1	20000	2	2	1	24	2	2	-1	-1	-2	...	0	0	0	0	68
2	120000	2	2	2	26	-1	2	0	0	0	...	3272	3455	3261	0	100
3	90000	2	2	2	34	0	0	0	0	0	...	14331	14948	15549	1518	150
4	50000	2	2	1	37	0	0	0	0	0	...	28314	28959	29547	2000	20
5	50000	1	2	1	57	-1	0	-1	0	0	...	20940	19146	19131	2000	3668
6	50000	1	1	2	37	0	0	0	0	0	...	19394	19619	20024	2500	18
7	500000	1	1	2	29	0	0	0	0	0	...	542653	483003	473944	55000	4000
8	100000	2	2	2	23	0	-1	-1	0	0	...	221	-159	567	380	60
9	140000	2	3	1	28	0	0	2	0	0	...	12211	11793	3719	3329	
10	20000	1	3	2	35	-2	-2	-2	-2	-1	...	0	13007	13912	0	

10 rows × 24 columns

In [134]:

```
d6.shape
```

Out[134]:

```
(30000, 24)
```

In [100]:

```
x_6 = d6.iloc[:,0:23].values.astype("int")
y_6 = d6.iloc[:,23].values.astype("int")
y_6
```

Out[100]:

```
array([1, 1, 0, ..., 1, 1, 1])
```

In [101]:

```
x6_train,x6_test,y6_train,y6_test = train_test_split(x_6,y_6,test_size = 0.33,random_state = 0)
x6_train2,x6_test2,y6_train2,y6_test2 = train_test_split(x_6,y_6,test_size = 0.33,random_state = 1)
x6_train3,x6_test3,y6_train3,y6_test3 = train_test_split(x_6,y_6,test_size = 0.33,random_state = 2)
x6_train4,x6_test4,y6_train4,y6_test4 = train_test_split(x_6,y_6,test_size = 0.33,random_state = 3)
x6_train5,x6_test5,y6_train5,y6_test5 = train_test_split(x_6,y_6,test_size = 0.33,random_state = 4)
```

In []:

In [144]:

```

#Decision Tree tuning
#from sklearn.tree import DecisionTreeClassifier
# tuning the model Decision Tree -1
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x5_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree__criterion=criterion,
                  dec_tree__max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x5_train, y5_train)
print("Trial 1")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree__criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree__max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca__n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

#Trial 2
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x5_train2.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree__criterion=criterion,
                  dec_tree__max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x5_train2, y5_train2)
print("Trial 2")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree__criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree__max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca__n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x5_train3.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree__criterion=criterion,
                  dec_tree__max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)

```

```

clf_GS.fit(x5_train3, y5_train3)
print("Trial 3")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x4_train5.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x5_train4, y5_train4)
print("Trial 4")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x5_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS.fit(x5_train5, y5_train5)
print("Trial 5")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

```

Trial 1

Best Criterion: entropy
Best max_depth: 40
Best Number Of Components: 13

DecisionTreeClassifier(criterion='entropy', max_depth=40)

Trial 2

Best Criterion: entropy
Best max_depth: 40
Best Number Of Components: 13

DecisionTreeClassifier(criterion='entropy', max_depth=40)

Trial 3

Best Criterion: entropy
Best max_depth: 45
Best Number Of Components: 14

DecisionTreeClassifier(criterion='entropy', max_depth=45)

Trial 4

Best Criterion: entropy
Best max_depth: 45
Best Number Of Components: 11

DecisionTreeClassifier(criterion='entropy', max_depth=45)

Trial 5

Best Criterion: entropy
Best max_depth: 40
Best Number Of Components: 11

DecisionTreeClassifier(criterion='entropy', max_depth=40)

In [142]:

```
#tuning random forest dataset6
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
rfc=RandomForestClassifier(random_state=1)
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x6_train, y6_train)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x6_train2, y6_train2)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x6_train3, y6_train3)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x6_train4, y6_train4)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x6_train5, y6_train5)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

{'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 500}
0.8174129353233832
```

In [139]:

```
#Trial 1
## Model - random forest -dataset 6
print("Trial 1")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 6,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x6_train,y6_train)
y6_pred = classifier.predict(x6_test)
print(confusion_matrix(y6_test,y6_pred))
print(classification_report(y6_test,y6_pred))
print(accuracy_score(y6_test,y6_pred))
print("ROC_AUC score:"+str(roc_auc_score(y6_test,y6_pred)))

print("Trial 2")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 7,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x6_train2,y6_train2)
y6_pred2 = classifier.predict(x6_test2)
print(confusion_matrix(y6_test2,y6_pred2))
print(classification_report(y6_test2,y6_pred2))
print(accuracy_score(y6_test2,y6_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y6_test2,y6_pred2)))

print("Trial 3")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 5,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x6_train3,y6_train3)
y6_pred3 = classifier.predict(x6_test3)
print(confusion_matrix(y6_test3,y6_pred3))
print(classification_report(y6_test3,y6_pred3))
print(accuracy_score(y6_test3,y6_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y6_test3,y6_pred3)))

print("Trial 4")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x6_train4,y6_train4)
y6_pred4 = classifier.predict(x6_test4)
print(confusion_matrix(y6_test4,y6_pred4))
print(classification_report(y6_test4,y6_pred4))
print(accuracy_score(y6_test4,y6_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y6_test4,y6_pred4)))

print("Trial 5")
classifier = RandomForestClassifier(criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 500)
```

```
classifier.fit(x6_train5,y6_train5)
y6_pred5 = classifier.predict(x6_test5)
print(confusion_matrix(y6_test5,y6_pred5))
print(classification_report(y6_test5,y6_pred5))
print(accuracy_score(y6_test5,y6_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y6_test5,y6_pred5)))
```

Trial 1

```
[[7461 304]
 [1472 663]]
```

	precision	recall	f1-score	support
0	0.84	0.96	0.89	7765
1	0.69	0.31	0.43	2135

accuracy		0.82	9900
----------	--	------	------

macro avg	0.76	0.64	0.66	9900
-----------	------	------	------	------

weighted avg	0.80	0.82	0.79	9900
--------------	------	------	------	------

0.8206060606060606

ROC_AUC score:0.6356943047452162

Trial 2

```
[[7345 345]
 [1458 752]]
```

	precision	recall	f1-score	support
0	0.83	0.96	0.89	7690
1	0.69	0.34	0.45	2210

accuracy		0.82	9900
----------	--	------	------

macro avg	0.76	0.65	0.67	9900
-----------	------	------	------	------

weighted avg	0.80	0.82	0.79	9900
--------------	------	------	------	------

0.8178787878787879

ROC_AUC score:0.6477040170874793

Trial 3

```
[[7430 276]
 [1531 663]]
```

	precision	recall	f1-score	support
0	0.83	0.96	0.89	7706
1	0.71	0.30	0.42	2194

accuracy		0.82	9900
----------	--	------	------

macro avg	0.77	0.63	0.66	9900
-----------	------	------	------	------

weighted avg	0.80	0.82	0.79	9900
--------------	------	------	------	------

0.8174747474747475

ROC_AUC score:0.6331857688938121

Trial 4

```
[[7437 349]
 [1349 765]]
```

	precision	recall	f1-score	support
0	0.85	0.96	0.90	7786
1	0.69	0.36	0.47	2114

accuracy		0.83	9900
----------	--	------	------

macro avg	0.77	0.66	0.69	9900
-----------	------	------	------	------

weighted avg	0.81	0.83	0.81	9900
--------------	------	------	------	------

0.8284848484848485

ROC_AUC score:0.6585245914786285

Trial 5

```
[[7323 355]
 [1462 760]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.83	0.95	0.89	7678
1	0.68	0.34	0.46	2222
accuracy			0.82	9900
macro avg	0.76	0.65	0.67	9900
weighted avg	0.80	0.82	0.79	9900

0.8164646464646464

ROC_AUC score: 0.6478991022311401

In [105]:

```

#Decision Tree
#from sklearn.tree import DecisionTreeClassifier
# Trial one
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=40)
classifier = classifier.fit(x6_train,y6_train)
y6_pred = classifier.predict(x6_test)
print("Trial 1")
print("Accuracy Score:",accuracy_score(y6_test,y6_pred))
print(classification_report(y6_test,y6_pred))
print("ROC_AUC score:"+str(roc_auc_score(y6_test,y6_pred)))

# Trial two
#from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=40)
classifier = classifier.fit(x6_train2,y6_train2)
y6_pred2 = classifier.predict(x6_test2)
print("Trial 2")
print("Accuracy Score:",accuracy_score(y6_test2,y6_pred2))
print(classification_report(y6_test2,y6_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y6_test2,y6_pred2)))
# Trial 3
#from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=40)
classifier = classifier.fit(x6_train3,y6_train3)
y6_pred3 = classifier.predict(x6_test3)
print("Trial 3")
print("Accuracy Score:",accuracy_score(y6_test3,y6_pred3))
print(classification_report(y6_test3,y6_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y6_test3,y6_pred3)))
# Trial 4
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=70)
classifier = classifier.fit(x6_train4,y6_train4)
y6_pred4 = classifier.predict(x6_test4)
print("Accuracy Score:",accuracy_score(y6_test4,y6_pred4))
print(classification_report(y6_test4,y6_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y6_test4,y6_pred4)))
# Trial 5
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=20)
classifier = classifier.fit(x6_train5,y6_train5)
y6_pred5 = classifier.predict(x6_test5)
print("Accuracy Score:",accuracy_score(y6_test5,y6_pred5))
print(classification_report(y6_test5,y6_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y6_test5,y6_pred5)))

```

Trial 1**Accuracy Score:** 0.7363636363636363

	precision	recall	f1-score	support
0	0.84	0.82	0.83	7765
1	0.39	0.41	0.40	2135
accuracy			0.74	9900
macro avg	0.62	0.62	0.62	9900
weighted avg	0.74	0.74	0.74	9900

ROC_AUC score: 0.6196875127237303**Trial 2****Accuracy Score:** 0.7327272727272728

	precision	recall	f1-score	support
0	0.83	0.82	0.83	7690
1	0.40	0.42	0.41	2210
accuracy			0.73	9900
macro avg	0.62	0.62	0.62	9900
weighted avg	0.74	0.73	0.73	9900

ROC_AUC score: 0.6198171216070704**Trial 3****Accuracy Score:** 0.7322222222222222

	precision	recall	f1-score	support
0	0.83	0.83	0.83	7706
1	0.40	0.40	0.40	2194
accuracy			0.73	9900
macro avg	0.61	0.61	0.61	9900
weighted avg	0.73	0.73	0.73	9900

ROC_AUC score: 0.614285391510859**Accuracy Score:** 0.7293939393939394

	precision	recall	f1-score	support
0	0.83	0.82	0.83	7786
1	0.38	0.40	0.39	2114
accuracy			0.73	9900
macro avg	0.61	0.61	0.61	9900
weighted avg	0.74	0.73	0.73	9900

ROC_AUC score: 0.6106893580185769**Accuracy Score:** 0.7566666666666667

	precision	recall	f1-score	support
0	0.83	0.86	0.85	7678
1	0.45	0.41	0.43	2222
accuracy			0.76	9900
macro avg	0.64	0.63	0.64	9900
weighted avg	0.75	0.76	0.75	9900

ROC_AUC score: 0.6317335888316625

Dataset 7

In [160]:

```
d7.head(5)
```

Out[160]:

	0	1	2	3	4	5	6	7	8
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	C
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	C
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	C
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	C
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	C

In [108]:

```
x_7 = d7.iloc[:,0:8].values.astype("int")
y_7 = d7.iloc[:,8].values.astype("int")
y_7
```

Out[108]:

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [109]:

```
x7_train,x7_test,y7_train,y7_test = train_test_split(x_7,y_7,test_size = 0.33,random_state = 0)
x7_train2,x7_test2,y7_train2,y7_test2 = train_test_split(x_7,y_7,test_size = 0.33,random_state = 1)
x7_train3,x7_test3,y7_train3,y7_test3 = train_test_split(x_7,y_7,test_size = 0.33,random_state = 2)
x7_train4,x7_test4,y7_train4,y7_test4 = train_test_split(x_7,y_7,test_size = 0.33,random_state = 3)
x7_train5,x7_test5,y7_train5,y7_test5 = train_test_split(x_7,y_7,test_size = 0.33,random_state = 4)
```

In [143]:

```
#tuning random forest dataset7
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
rfc=RandomForestClassifier(random_state=1)
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x7_train, y7_train)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x7_train2, y7_train2)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x7_train3, y7_train3)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x7_train4, y7_train4)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x7_train5, y7_train5)
print(CV_rfc.best_params_)
print(CV_rfc.best_score_)

{'criterion': 'entropy', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 200}
0.9774831464736662
```

In [145]:

```

#Decision Tree tuning
#from sklearn.tree import DecisionTreeClassifier
# tuning the model Decision Tree -1
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x7_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree__criterion=criterion,
                  dec_tree__max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x7_train, y7_train)
print("Trial 1")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree__criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree__max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca__n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

#Trial 2
std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x7_train2.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree__criterion=criterion,
                  dec_tree__max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x7_train2, y7_train2)
print("Trial 2")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree__criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree__max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca__n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x7_train3.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca__n_components=n_components,
                  dec_tree__criterion=criterion,
                  dec_tree__max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)

```

```

clf_GS.fit(x7_train3, y7_train3)
print("Trial 3")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x7_train5.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(x7_train4, y7_train4)
print("Trial 4")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

std_slc = StandardScaler()
pca = decomposition.PCA()
dec_tree = tree.DecisionTreeClassifier()
pipe = Pipeline(steps = [("std_slc",std_slc),("pca",pca),("dec_tree",dec_tree)])
n_components = list(range(1,x7_train.shape[1]+1,1))
criterion = ['gini','entropy']
max_depth = [40,45,60]
parameters = dict(pca_n_components=n_components,
                  dec_tree_criterion=criterion,
                  dec_tree_max_depth=max_depth)
clf_GS.fit(x7_train5, y7_train5)
print("Trial 5")
print('Best Criterion:', clf_GS.best_estimator_.get_params()['dec_tree_criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dec_tree_max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca_n_components'])
print(); print(clf_GS.best_estimator_.get_params()['dec_tree'])

```

```
Trial 1
```

```
Best Criterion: entropy  
Best max_depth: 45  
Best Number Of Components: 5
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=45)
```

```
Trial 2
```

```
Best Criterion: entropy  
Best max_depth: 60  
Best Number Of Components: 3
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=60)
```

```
Trial 3
```

```
Best Criterion: entropy  
Best max_depth: 45  
Best Number Of Components: 7
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=45)
```

```
Trial 4
```

```
Best Criterion: entropy  
Best max_depth: 60  
Best Number Of Components: 4
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=60)
```

```
Trial 5
```

```
Best Criterion: entropy  
Best max_depth: 60  
Best Number Of Components: 6
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=60)
```

```
In [ ]:
```

```
In [ ]:
```

In [111]:

```

#Decision Tree
#from sklearn.tree import DecisionTreeClassifier
#Trial 1
#Decision Tree
#from sklearn.tree import DecisionTreeClassifier
# Trial one
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=40)
classifier = classifier.fit(x7_train,y7_train)
y7_pred = classifier.predict(x7_test)
print("Trial 1")
print("Accuracy Score:",accuracy_score(y7_test,y7_pred))
print(classification_report(y7_test,y7_pred))
print("ROC_AUC score:"+str(roc_auc_score(y7_test,y7_pred)))

# Trial two
#from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=40)
classifier = classifier.fit(x7_train2,y7_train2)
y7_pred2 = classifier.predict(x7_test2)
print("Trial 2")
print("Accuracy Score:",accuracy_score(y7_test2,y7_pred2))
print(classification_report(y7_test2,y7_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y7_test2,y7_pred2)))
# Trial 3
#from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=40)
classifier = classifier.fit(x7_train3,y7_train3)
y7_pred3 = classifier.predict(x7_test3)
print("Trial 3")
print("Accuracy Score:",accuracy_score(y7_test3,y7_pred3))
print(classification_report(y7_test3,y7_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y7_test3,y7_pred3)))
# Trial 4
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=50)
classifier = classifier.fit(x7_train4,y7_train4)
y7_pred4 = classifier.predict(x7_test4)
print("Accuracy Score:",accuracy_score(y7_test4,y7_pred4))
print(classification_report(y7_test4,y7_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y7_test4,y7_pred4)))
# Trial 5
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=45)
classifier = classifier.fit(x7_train5,y7_train5)
y7_pred5 = classifier.predict(x7_test5)
print("Accuracy Score:",accuracy_score(y7_test5,y7_pred5))
print(classification_report(y7_test5,y7_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y7_test5,y7_pred5)))

```

Trial 1**Accuracy Score:** 0.9620788894531911

	precision	recall	f1-score	support
0	0.98	0.98	0.98	5429
1	0.74	0.81	0.78	478
accuracy			0.96	5907
macro avg	0.86	0.89	0.88	5907
weighted avg	0.96	0.96	0.96	5907

ROC_AUC score: 0.8944705367347677**Trial 2****Accuracy Score:** 0.9608938547486033

	precision	recall	f1-score	support
0	0.98	0.98	0.98	5374
1	0.77	0.80	0.79	533
accuracy			0.96	5907
macro avg	0.88	0.89	0.88	5907
weighted avg	0.96	0.96	0.96	5907

ROC_AUC score: 0.8897778268097875**Trial 3****Accuracy Score:** 0.9634332148298629

	precision	recall	f1-score	support
0	0.98	0.98	0.98	5364
1	0.80	0.81	0.80	543
accuracy			0.96	5907
macro avg	0.89	0.89	0.89	5907
weighted avg	0.96	0.96	0.96	5907

ROC_AUC score: 0.8946233535623205**Accuracy Score:** 0.9610631454206873

	precision	recall	f1-score	support
0	0.98	0.98	0.98	5370
1	0.78	0.80	0.79	537
accuracy			0.96	5907
macro avg	0.88	0.89	0.88	5907
weighted avg	0.96	0.96	0.96	5907

ROC_AUC score: 0.8880819366852886**Accuracy Score:** 0.9642796681902828

	precision	recall	f1-score	support
0	0.98	0.98	0.98	5371
1	0.79	0.82	0.81	536
accuracy			0.96	5907
macro avg	0.89	0.90	0.89	5907
weighted avg	0.96	0.96	0.96	5907

ROC_AUC score: 0.8989023764995541

In [140]:

```
#Trial 1
## Model - random forest -dataset 7
print("Trial 1")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 6,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x7_train,y7_train)
y7_pred = classifier.predict(x7_test)
print(confusion_matrix(y7_test,y7_pred))
print(classification_report(y7_test,y7_pred))
print(accuracy_score(y7_test,y7_pred))
print("ROC_AUC score:"+str(roc_auc_score(y7_test,y7_pred)))

print("Trial 2")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 7,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x7_train2,y7_train2)
y7_pred2 = classifier.predict(x7_test2)
print(confusion_matrix(y7_test2,y7_pred2))
print(classification_report(y7_test2,y7_pred2))
print(accuracy_score(y7_test2,y7_pred2))
print("ROC_AUC score:"+str(roc_auc_score(y7_test2,y7_pred2)))

print("Trial 3")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 5,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x7_train3,y7_train3)
y7_pred3 = classifier.predict(x7_test3)
print(confusion_matrix(y7_test3,y7_pred3))
print(classification_report(y7_test3,y7_pred3))
print(accuracy_score(y7_test3,y7_pred3))
print("ROC_AUC score:"+str(roc_auc_score(y7_test3,y7_pred3)))

print("Trial 4")
classifier = RandomForestClassifier(
    criterion = 'gini',
    max_depth = 7,
    max_features = 'auto',
    n_estimators = 500)
classifier.fit(x7_train4,y7_train4)
y7_pred4 = classifier.predict(x7_test4)
print(confusion_matrix(y7_test4,y7_pred4))
print(classification_report(y7_test4,y7_pred4))
print(accuracy_score(y7_test4,y7_pred4))
print("ROC_AUC score:"+str(roc_auc_score(y7_test4,y7_pred4)))

print("Trial 5")
classifier = RandomForestClassifier(criterion = 'gini',
    max_depth = 8,
    max_features = 'auto',
    n_estimators = 500)
```

```
classifier.fit(x7_train5,y7_train5)
y6_pred5 = classifier.predict(x7_test5)
print(confusion_matrix(y7_test5,y7_pred5))
print(classification_report(y7_test5,y7_pred5))
print(accuracy_score(y7_test5,y7_pred5))
print("ROC_AUC score:"+str(roc_auc_score(y7_test5,y7_pred5)))
```

Trial 1

```
[[5390  39]
 [ 84  394]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.98	0.99	0.99	5429
1	0.91	0.82	0.86	478

accuracy			0.98	5907
----------	--	--	------	------

macro avg	0.95	0.91	0.93	5907
weighted avg	0.98	0.98	0.98	5907

0.979177247333672

ROC_AUC score:0.9085420695151022

Trial 2

```
[[5340  34]
 [ 97  436]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.98	0.99	0.99	5374
1	0.93	0.82	0.87	533

accuracy			0.98	5907
----------	--	--	------	------

macro avg	0.95	0.91	0.93	5907
weighted avg	0.98	0.98	0.98	5907

0.9778229219570002

ROC_AUC score:0.9058422492844778

Trial 3

```
[[5319  45]
 [ 93  450]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.98	0.99	0.99	5364
1	0.91	0.83	0.87	543

accuracy			0.98	5907
----------	--	--	------	------

macro avg	0.95	0.91	0.93	5907
weighted avg	0.98	0.98	0.98	5907

0.9766378872524124

ROC_AUC score:0.9101700100114947

Trial 4

```
[[5330  40]
 [104  433]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.98	0.99	0.99	5370
1	0.92	0.81	0.86	537

accuracy			0.98	5907
----------	--	--	------	------

macro avg	0.95	0.90	0.92	5907
weighted avg	0.97	0.98	0.97	5907

0.9756221432199086

ROC_AUC score:0.8994413407821229

Trial 5

```
[[5257 114]
 [ 97  439]]
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.98	0.98	0.98	5371
1	0.79	0.82	0.81	536
accuracy			0.96	5907
macro avg	0.89	0.90	0.89	5907
weighted avg	0.96	0.96	0.96	5907

0.9642796681902828

ROC_AUC score: 0.8989023764995541

In []:

In []:

In []:

In [1]:

```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import LinearSVC
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectFromModel
```

In [2]:

```
# dataset1
electrical = pd.read_csv("Data_for_UCI_named(1).csv")
electrical = electrical.replace(to_replace='unstable', value = 0)
electrical = electrical.replace(to_replace='stable', value = 1)
electricalX = electrical.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,11]]
electricalY = electrical['stabf']
```

In [3]:

```
#on dataset 1
X1_train, X1_test, Y1_train, Y1_test = train_test_split(electricalX, electricalY
, test_size=0.33,
                                         shuffle=True,random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(electricalX, electricalY
, test_size=0.33,
                                         shuffle=True,random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(electricalX, electricalY
, test_size=0.33,
                                         shuffle=True,random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(electricalX, electricalY
, test_size=0.33,
                                         shuffle=True,random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(electricalX, electricalY
, test_size=0.33,
                                         shuffle=True,random_state=5)
```

In [4]:

```
print(X1_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1',dual=False).fit(X1_train,Y1_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X1_train)
print(X_new.shape)

(6700, 12)
(6700, 9)
```

In [5]:

```
print(X2_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X2_train, Y2_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X2_train)
print(X_new.shape)
```

```
(6700, 12)
(6700, 9)
```

In [6]:

```
print(X3_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X3_train, Y3_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X3_train)
print(X_new.shape)
```

```
(6700, 12)
(6700, 9)
```

In [7]:

```
print(X4_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X4_train, Y4_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X4_train)
print(X_new.shape)
```

```
(6700, 12)
(6700, 9)
```

In [8]:

```
print(X5_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X5_train, Y5_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X5_train)
print(X_new.shape)
```

```
(6700, 12)
(6700, 9)
```

In [9]:

```
# dataset2
AI4I = pd.read_csv("ai4i2020(1).csv")
AI4I = AI4I.replace(to_replace='M', value = 0)
AI4I = AI4I.replace(to_replace='L', value = 1)
AI4I = AI4I.replace(to_replace='H', value = 2)
AI4IX = AI4I.iloc[:,[2,3,4,5,6,7]]
AI4IY = AI4I['Machine failure']
```

In [10]:

```
#on dataset 2
X1_train, X1_test, Y1_train, Y1_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(AI4IX, AI4IY, test_size=0.33,
                                                       shuffle=True, random_state=5)
```

In [11]:

```
print(X1_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X1_train, Y1_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X1_train)
print(X_new.shape)
```

```
(6700, 6)
(6700, 5)
```

```
C:\Users\dugur\anaconda3\lib\site-packages\sklearn\svm\_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
    warnings.warn("Liblinear failed to converge, increase "
```

In [12]:

```
print(X2_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X2_train, Y2_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X2_train)
print(X_new.shape)
```

```
(6700, 6)
(6700, 5)
```

```
C:\Users\dugur\anaconda3\lib\site-packages\sklearn\svm\_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
    warnings.warn("Liblinear failed to converge, increase "
```

In [13]:

```
print(X3_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X3_train, Y3_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X3_train)
print(X_new.shape)
```

(6700, 6)
(6700, 5)

C:\Users\dugur\anaconda3\lib\site-packages\sklearn\svm_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.
warnings.warn("Liblinear failed to converge, increase "

In [14]:

```
print(X4_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X4_train, Y4_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X4_train)
print(X_new.shape)
```

(6700, 6)
(6700, 5)

C:\Users\dugur\anaconda3\lib\site-packages\sklearn\svm_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.

warnings.warn("Liblinear failed to converge, increase "

In [15]:

```
print(X5_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X5_train, Y5_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X5_train)
print(X_new.shape)
```

(6700, 6)
(6700, 5)

C:\Users\dugur\anaconda3\lib\site-packages\sklearn\svm_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.

warnings.warn("Liblinear failed to converge, increase "

In [16]:

```
#dataset3
occup_1 = pd.read_csv("datatraining(1).csv")
occup_2 = pd.read_csv("datatest(1).csv")
occup = occup_1.append(occup_2)
occupX = occup.iloc[:, [1, 2, 3, 4, 5]]
occupY = occup['Occupancy']
```

In [17]:

```
#on dataset 3
X1_train, X1_test, Y1_train, Y1_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(occupX, occupY, test_size=0.33,
                                                       shuffle=True, random_state=5)
```

In [18]:

```
print(X1_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X1_train, Y1_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X1_train)
print(X_new.shape)

(7241, 5)
(7241, 4)
```

In [19]:

```
print(X2_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X2_train, Y2_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X2_train)
print(X_new.shape)

(7241, 5)
(7241, 4)
```

In [20]:

```
print(X3_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X3_train, Y3_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X3_train)
print(X_new.shape)

(7241, 5)
(7241, 4)
```

In [21]:

```
print(X4_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X4_train, Y4_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X4_train)
print(X_new.shape)

(7241, 5)
(7241, 4)
```

In [22]:

```
print(X5_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X5_train, Y5_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X5_train)
print(X_new.shape)

(7241, 5)
(7241, 4)
```

In [23]:

```
#dataset4
internet = pd.read_csv("log2.csv")
internet = internet.replace(to_replace='allow', value = 1)
internet = internet.replace(to_replace='drop', value = 0)
internet = internet.replace(to_replace='deny', value = 0)
internet = internet.replace(to_replace='reset-both', value = 0)
internetX = internet.iloc[:,[5,6,7,8,9]]
internety = internet['Action']
```

In [24]:

```
#on dataset 4
X1_train, X1_test, Y1_train, Y1_test = train_test_split(internetX, internety, test_size=0.33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(internetX, internety, test_size=0.33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(internetX, internety, test_size=0.33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(internetX, internety, test_size=0.33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(internetX, internety, test_size=0.33,
                                                       shuffle=True, random_state=5)
```

In [25]:

```
print(X1_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X1_train, Y1_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X1_train)
print(X_new.shape)
```

(43906, 5)
(43906, 5)

In [26]:

```
print(X2_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X2_train, Y2_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X2_train)
print(X_new.shape)
```

(43906, 5)
(43906, 5)

In [27]:

```
print(X3_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X3_train, Y3_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X3_train)
print(X_new.shape)
```

(43906, 5)
(43906, 5)

In [28]:

```
print(X4_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X4_train, Y4_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X4_train)
print(X_new.shape)
```

(43906, 5)
(43906, 5)

In [29]:

```
print(X5_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X5_train, Y5_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X5_train)
print(X_new.shape)
```

(43906, 5)
(43906, 5)

In [30]:

```
#dataset5
col_names = ["AF3", "F7", "F3", "FC5", "T7", "P7", "O1", "O2", "P8", "T8", "FC6",
              "F4", "F8", "AF4", "eyeDetection01"]
eeg = pd.read_csv("EEG.csv", names=col_names)
eeg = eeg.drop([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16])
eegX = eeg.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13]]
eegY = eeg['eyeDetection01']
```

In [31]:

```
#on dataset 5
X1_train, X1_test, Y1_train, Y1_test = train_test_split(eegX, eegY, test_size=0.
33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(eegX, eegY, test_size=0.
33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(eegX, eegY, test_size=0.
33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(eegX, eegY, test_size=0.
33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(eegX, eegY, test_size=0.
33,
                                                       shuffle=True, random_state=5)
```

In [32]:

```
print(X1_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X1_train, Y1_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X1_train)
print(X_new.shape)

(10036, 14)
(10036, 11)

C:\Users\dugur\anaconda3\lib\site-packages\sklearn\svm\_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

In [33]:

```
print(X2_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X2_train, Y2_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X2_train)
print(X_new.shape)
```

(10036, 14)
(10036, 13)

C:\Users\dugur\anaconda3\lib\site-packages\sklearn\svm_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.
warnings.warn("Liblinear failed to converge, increase "

In [34]:

```
print(X3_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X3_train, Y3_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X3_train)
print(X_new.shape)
```

(10036, 14)
(10036, 12)

C:\Users\dugur\anaconda3\lib\site-packages\sklearn\svm_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.

warnings.warn("Liblinear failed to converge, increase "

In [35]:

```
print(X4_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X4_train, Y4_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X4_train)
print(X_new.shape)
```

(10036, 14)
(10036, 12)

C:\Users\dugur\anaconda3\lib\site-packages\sklearn\svm_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.

warnings.warn("Liblinear failed to converge, increase "

In [36]:

```
print(X5_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X5_train, Y5_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X5_train)
print(X_new.shape)
```

```
(10036, 14)
(10036, 12)
```

```
C:\Users\dugur\anaconda3\lib\site-packages\sklearn\svm\_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
warnings.warn("Liblinear failed to converge, increase "
```

In [37]:

```
#dataset6
credit = pd.read_excel("credit.xls", index_col=0)
credit = credit.drop('ID', axis=0)
credit = credit.values
creditX = credit[:, :23]
creditY = credit[:, 23]
```

In [38]:

```
#on dataset 6
X1_train, X1_test, Y1_train, Y1_test = train_test_split(creditX, creditY, test_size=0.33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(creditX, creditY, test_size=0.33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(creditX, creditY, test_size=0.33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(creditX, creditY, test_size=0.33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(creditX, creditY, test_size=0.33,
                                                       shuffle=True, random_state=5)
```

In [40]:

```
print(X1_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X1_train.astype(int),
                                                       Y1_train.astype(int))
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X1_train.astype(int))
print(X_new.shape)

(20100, 23)
(20100, 9)
```

In [42]:

```
print(X2_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X2_train.astype(int),
                                                       Y2_train.astype(int))
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X2_train.astype(int))
print(X_new.shape)
```

(20100, 23)
(20100, 10)

In [43]:

```
print(X3_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X3_train.astype(int),
                                                       Y3_train.astype(int))
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X3_train.astype(int))
print(X_new.shape)
```

(20100, 23)
(20100, 10)

In [44]:

```
print(X4_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X4_train.astype(int),
                                                       Y4_train.astype(int))
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X4_train.astype(int))
print(X_new.shape)
```

(20100, 23)
(20100, 9)

In [45]:

```
print(X5_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X5_train.astype(int),
                                                       Y5_train.astype(int))
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X5_train.astype(int))
print(X_new.shape)
```

(20100, 23)
(20100, 10)

In [46]:

```
#dataset7
htru = pd.read_csv("HTRU_2.csv", header=None)
htruX = htru.iloc[:,[0,1,2,3,4,5,6,7]]
htruY = htru.iloc[:,8]
```

In [47]:

```
#on dataset 7
X1_train, X1_test, Y1_train, Y1_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True, random_state=1)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True, random_state=2)
X3_train, X3_test, Y3_train, Y3_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True, random_state=3)
X4_train, X4_test, Y4_train, Y4_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True, random_state=4)
X5_train, X5_test, Y5_train, Y5_test = train_test_split(htruX, htruY, test_size=0.33,
                                                       shuffle=True, random_state=5)
```

In [48]:

```
print(X1_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X1_train, Y1_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X1_train)
print(X_new.shape)

(11991, 8)
(11991, 7)
```

In [49]:

```
print(X2_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X2_train, Y2_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X2_train)
print(X_new.shape)

(11991, 8)
(11991, 8)
```

In [50]:

```
print(X3_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X3_train, Y3_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X3_train)
print(X_new.shape)

(11991, 8)
(11991, 8)
```

In [51]:

```
print(X4_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X4_train, Y4_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X4_train)
print(X_new.shape)
```

```
(11991, 8)
(11991, 8)
```

In [52]:

```
print(X5_train.shape)
lsvc = LinearSVC(C=0.01, penalty='l1', dual=False).fit(X5_train, Y5_train)
model = SelectFromModel(lsvc, prefit=True)
X_new = model.transform(X5_train)
print(X_new.shape)
```

```
(11991, 8)
(11991, 8)
```

In []: