McMaster University

Comparing Classifiers Project

SFWRTECH 4DM3 - Data Mining

Aiden WangYang Li – 400374502
Xiran Dong – 400346507

4-1-2023

Introduction

In this project, we applied K-Nearest Neighbour, Naïve Bayes, AdaBoost, and Support Vector Machine classifiers to accelerometer data for classification tasks. Then we compared the results generated from those four classifiers. The computational times, cross validation analysis, receiver operating characteristic (ROC) curves, and confusion matrix are recorded for each classification results. This report analyzes the performance and parameters of each classifier to compare the effect and difference of each.

Code Execution

Our program is written in Python programming language version 3.10. To execute the code, please open it with any compiler available to run pythons such as Visual Studio Code or PyCharm. Here are the steps that can guide you to the results calculated by the program.

1. Open the main.py file by using any compiler available to run python.
2. Run the main.py without debug mode.
3. The result will be displayed on the terminal screen.

Accelerometer Dataset

The accelerometer dataset was generated from the vibrations of a cooler fan with weights on its blades. It can be used for predictions, classification and other tasks that require vibration analysis, especially in engines. (UCI Machine Learning Repository: Accelerometer Data Set, n.d.) [1] The dataset has 153000 instances. It has 3 classes and 4 attributes to categorize data. In this project, we focus on classes 1 and 2, and the attributes of x and y values. Class 1 and Class 2 have 51000 instances on each dataset, we took 38250 (75%) instances for training and validation, 12750 (25%) instances for testing.

K-Nearest Neighbours (KNN)

The K-Nearest Neighbors (KNN) algorithm is a type of supervised machine learning algorithm used for classification and regression analysis. In KNN, a data point is classified by comparing it with its nearest neighbors. Specifically, the KNN algorithm identifies the K data points in the training dataset that are closest to the test data point, and then assigns the test data point to the class that is most common among those K neighbors.

We have used 75% instances for training and 25% instances for testing to get the best K value from the KNN algorithm provided from Sklearn. From about 40 minutes execution of codes, we got the K value equals 279 with the error rate of 37.38% as the best result. The result could be overfitting because we have only one training set to train the data.

[1] UCI Machine Learning Repository: Accelerometer Data Set. (n.d.). http://archive.ics.uci.edu/ml/datasets/Accelerometer

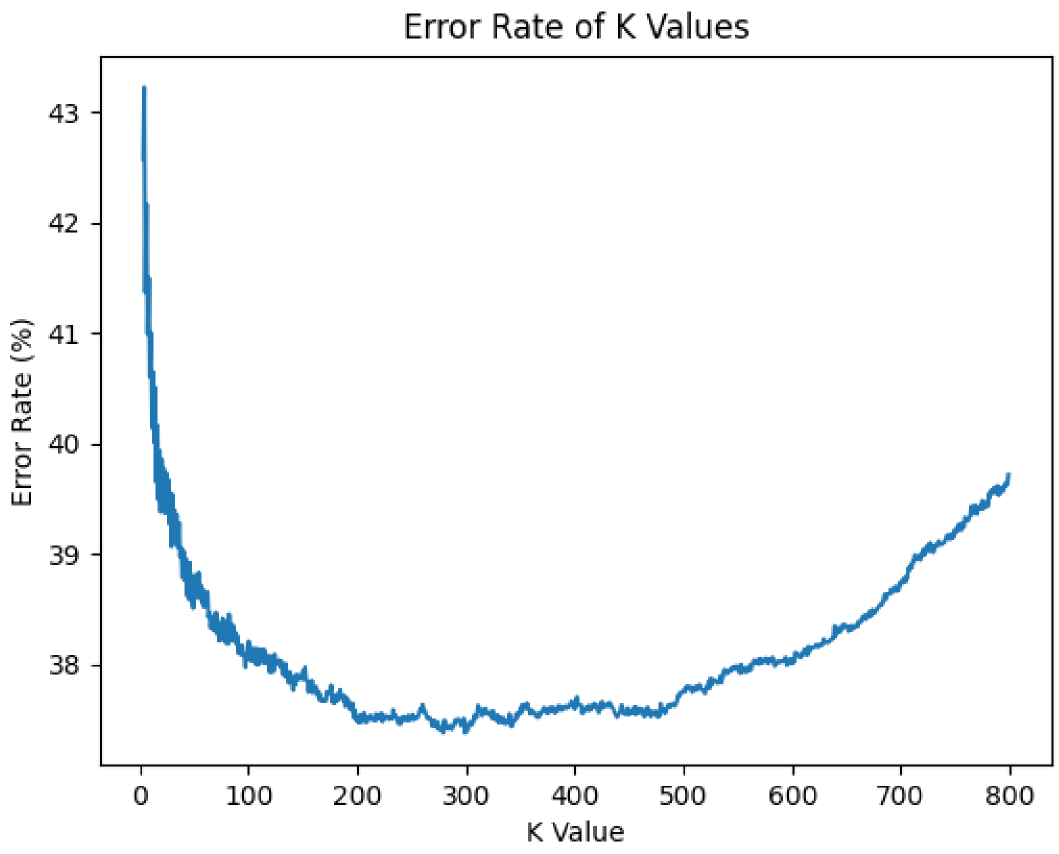

Figure KNN 1 - Best K value is 279 with error rate of 37.38%.

Computational Times of KNN

From the results of the code, we recorded the computation time for the K-Nearest Neighbour algorithm by using Sklearn as follows:

- Computational times for training data is 76.81 milliseconds.
- Computational times for testing data is 2238.01 milliseconds.

Confusion Matrix of KNN

Based on the results generated from the testing dataset using the K-Nearest Neighbour algorithm, we can plot the confusion matrix as follows: Overall, Error rate is 37.38%.

Class B misclassified as Class A (**False positive**) **5179 (40.62%)**	Class A correctly classified (**True positive**) **8398 (65.87%)**
Class B correctly classified (**True negative**) **7571 (59.38%)**	Class A misclassified as Class B (**False negative**) **4352 (34.13%)**

Cross Validation of KNN

Cross-validation is applied to evaluate the performance of using KNN on the accelerometer dataset. We have used the 4-fold cross validation method to find the best K values from the training set and validation set. The graph report below shows the brief trend of the error rate on increasing K values from 3 to 800. The reports are generated as four excel files called "KNN Cross Validation Partition # Data.xlsx".

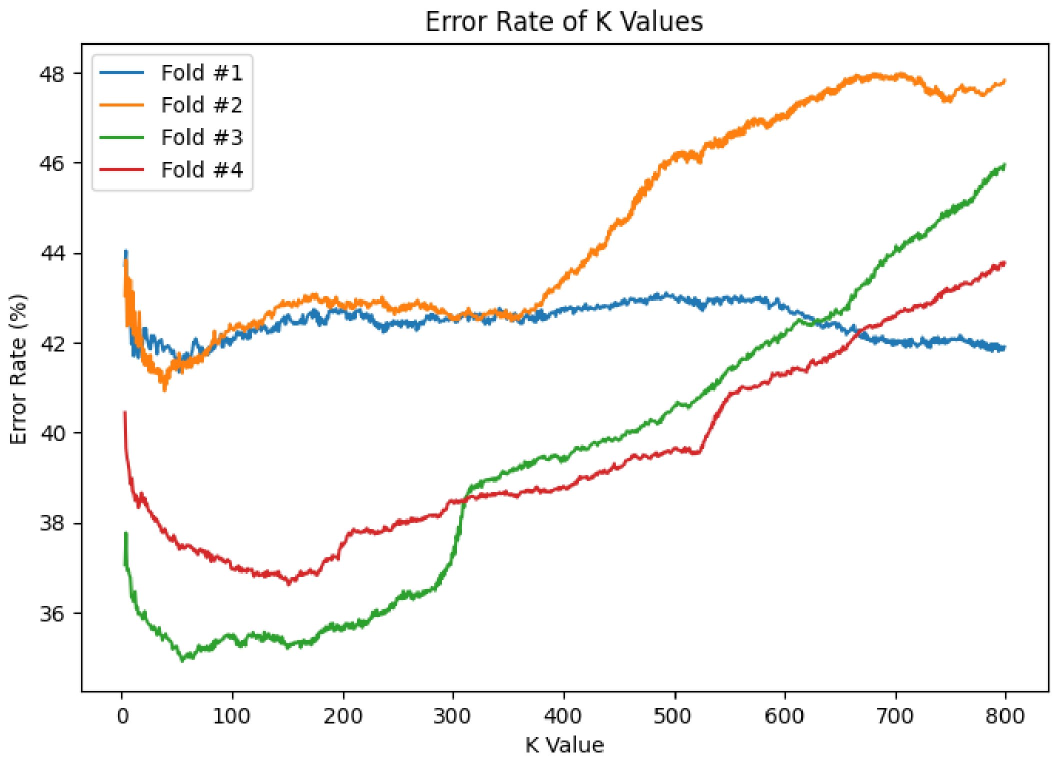

Figure KNN 2 – Error Rate of Each K values by using 4-Fold Cross Validation

In the code, uncomment and execute the *KNN_cross_validation ()* function to get the result. It took about 2 hours to get the results.

Cross Validation Results of KNN

- Fold #1: Best K value is 52 with Error rate of 41.35%
- Fold #2: Best K value is 39 with Error rate of 40.93%
- Fold #3: Best K value is 55 with Error rate of 34.91%
- Fold #4: Best K value is 151 with Error rate of 36.61%
- Average Error Rate is 38.45%

Receiver Operating Characteristic (ROC) Curve of KNN

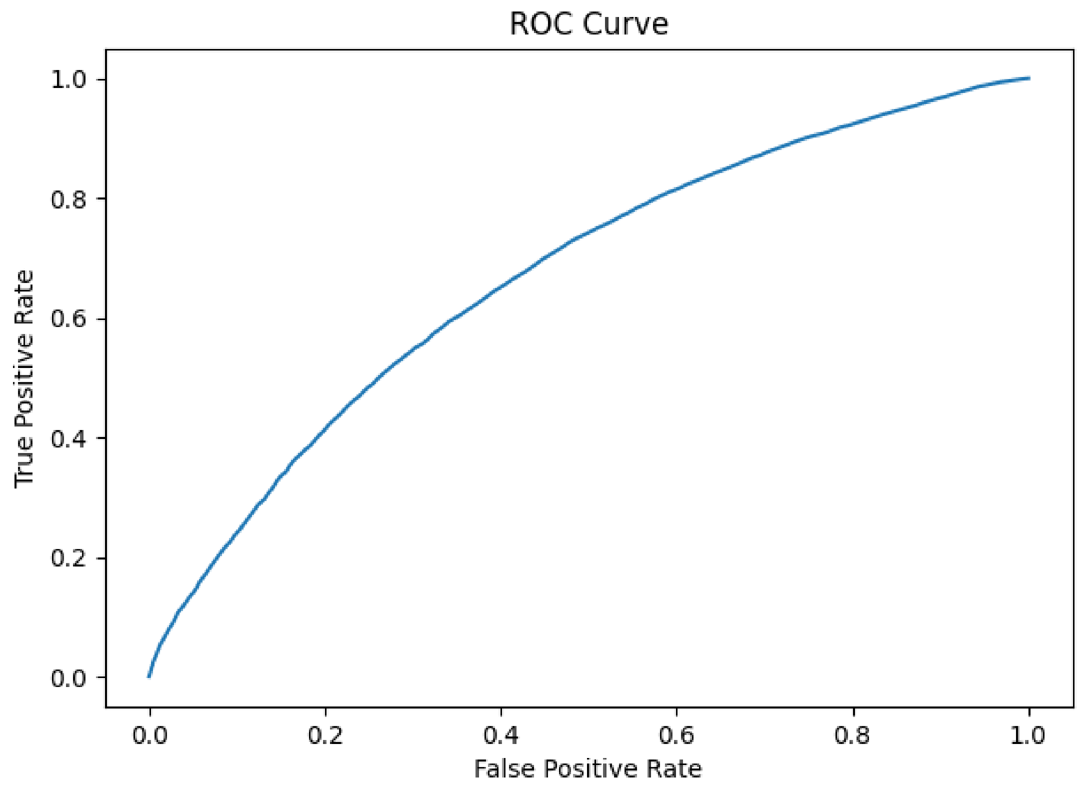

Figure KNN 3 – ROC Curve of the KNN Algorithm

Through the Receiver Operating Characteristic (ROC) calculation provided by Sklearn, we got the above graph. The positive class we set is 2. From this graph, we can see that the overall curve tends to be positive, and the result is not particularly good.

Gaussian Naive Bayes (GNB)

Gaussian Naive Bayes is a probabilistic machine learning algorithm used for classification tasks. It is based on Bayes' theorem, which states that the probability of a hypothesis is updated based on the evidence. In Gaussian Naive Bayes, we assume that the likelihood of each feature given the class label follows a Gaussian distribution (also known as a normal distribution). This means that the probability of observing a particular value of a feature, given the class label, can be calculated using the mean and standard deviation of the feature values for that class.

So, in this case, we do not need to determine the best value for training and testing data like KNN did.

Computational Times of GNB

From the results of the code, we recorded the computation time for the Gaussian Naive Bayes algorithm by using Sklearn as follows:

- Computational Times for Training data is 11.99 milliseconds.
- Computational Times for Testing data is 6.01 milliseconds.

Confusion Matrix of GNB

Based on the results generated from the testing dataset using the Gaussian Naive Bayes algorithm, we can plot the confusion matrix as follows: Overall, Error rate is 48.55%.

Class B misclassified as Class A (**False positive**) **12008 (94.18%)**	Class A correctly classified (**True positive**) **12377 (97.07%)**
Class B correctly classified (**True negative**) **742 (5.82%)**	Class A misclassified as Class B (**False negative**) **373 (2.93%)**

Cross Validation of GNB

Cross-validation is applied to evaluate the performance of using GNB on the accelerometer dataset. We have used the 4-fold cross validation method from the training set and validation set. A report is generated called "GNB Cross Validation Partition Data.xlsx".

In the code, uncomment and execute the *GNB_cross_validation ()* function to get the result. It took about few seconds to get the results.

Cross Validation Results of GNB

- Fold #1: Error rate is 50.00%
- Fold #2: Error rate is 49.94%
- Fold #3: Error rate is 45.90%
- Fold #4: Error rate is 49.19%
- Average Error Rate is 48.76%

Receiver Operating Characteristic (ROC) Curve of KNN

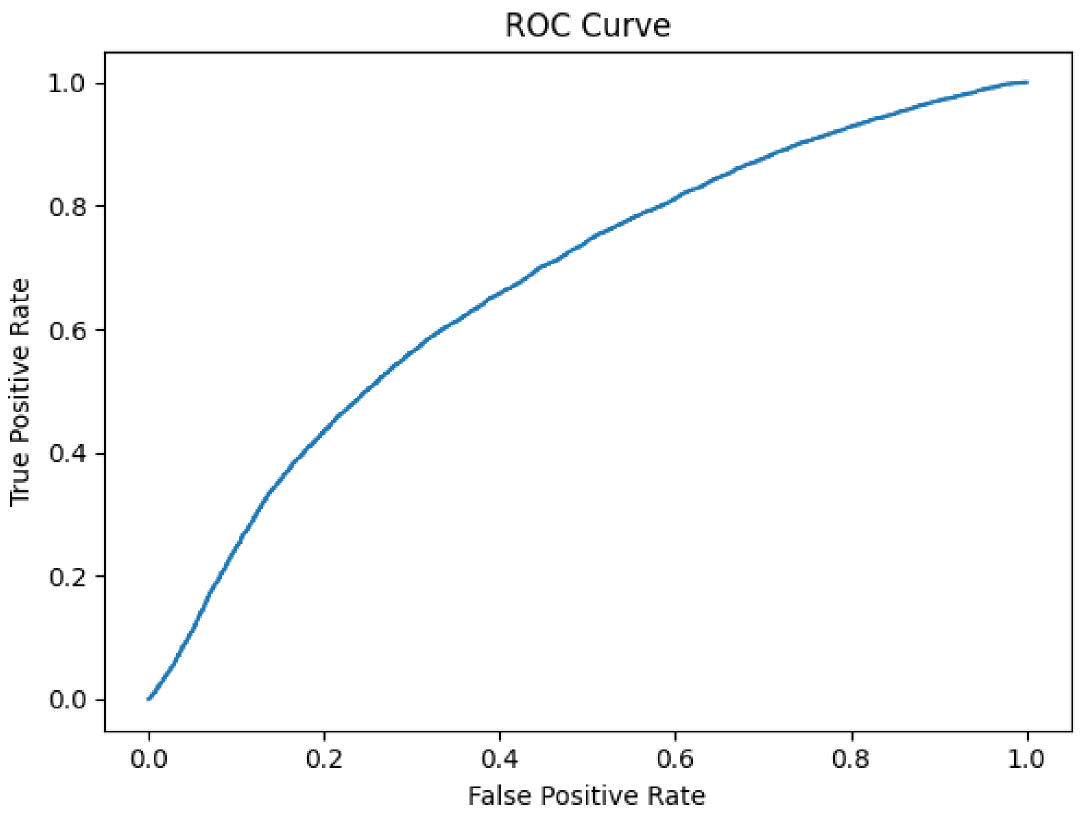

Figure GNB 1 – ROC Curve of the GNB Algorithm

Through the Receiver Operating Characteristic (ROC) calculation provided by Sklearn, we got the above graph. The positive class we set is 2. From this graph, we can see that the overall curve tends to be positive, and the result is not particularly good.

Adaptive Boosting (AdaBoost)

Adaptive Boosting (AdaBoost) is a machine learning algorithm that combines multiple weak classifiers to create a strong classifier. The term "weak classifier" refers to a classifier that performs only slightly better than random guessing. The idea behind AdaBoost is to iteratively train a series of weak classifiers on the data and assign weights to the training samples based on their classification accuracy. The weights of misclassified samples are increased, so that the next weak classifier focuses more on those samples in the training process. In contrast, the weights of correctly classified samples are decreased, so that the next weak classifier focuses less on those samples. After training a series of weak classifiers, AdaBoost combines them into a strong classifier by weighting their predictions based on their individual accuracy. The final prediction is a weighted sum of the weak classifier predictions, where the weights are determined by the accuracy of each weak classifier.

We have used 75% instances for training and 25% instances for testing to get the best Estimator value from the AdaBoost algorithm provided from Sklearn. From about 2 minutes execution of codes, we got the Estimator value equals 16 with the error rate of 40.71% as the best result. The result could be overfitting because we have only one training set to train the data.

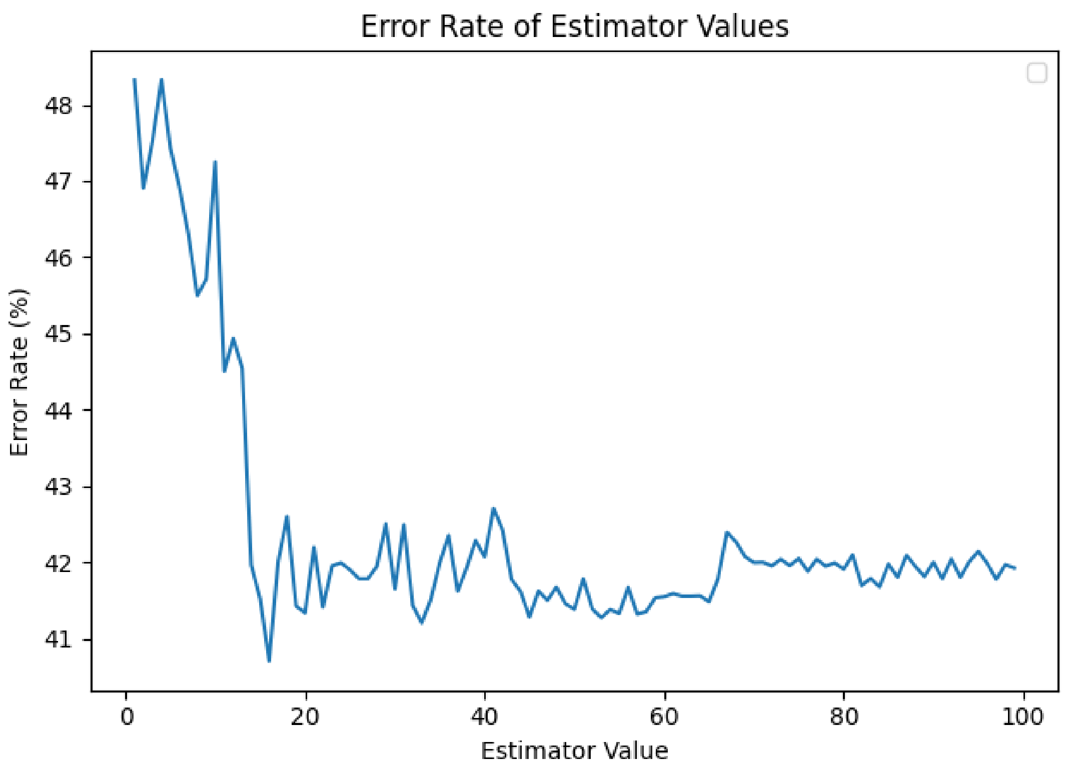

Figure ADB 1 - Best Estimator value is 16 with error rate of 40.71%

Computational Times of AdaBoost

From the results of the code, we recorded the computation time for the Adaptive Boosting algorithm by using Sklearn as follows:

- Computational Times for Training data is 385.99 milliseconds.
- Computational Times for Testing data is 59.87 milliseconds.

Confusion Matrix of AdaBoost

Based on the results generated from the testing dataset using the Gaussian Naive Bayes algorithm, we can plot the confusion matrix as follows: Overall, Error rate is 40.71%

Class B misclassified as Class A (**False positive**) **5322 (41.74%)**	Class A correctly classified (**True positive**) **7691 (60.32%)**
Class B correctly classified (**True negative**) **7428 (58.26%)**	Class A misclassified as Class B (**False negative**) **5059 (39.68%)**

Cross Validation of AdaBoost

Cross-validation is applied to evaluate the performance of using AdaBoost on the accelerometer dataset. We have used the 4-fold cross validation method to find the best Estimator values from the training set and validation set. The graph report below shows the brief trend of the error rate on increasing Estimator values from 1 to 99. The reports are generated as four excel files called "AdaBoost Cross Validation Partition # Data.xlsx

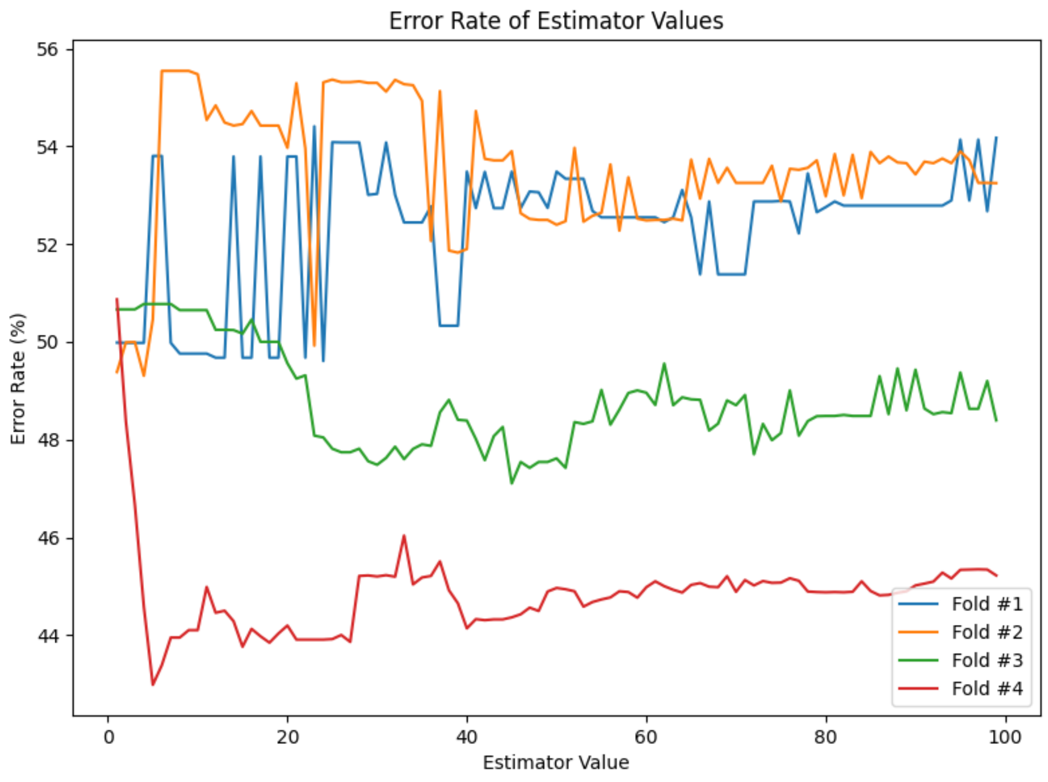

Figure ADB 2 – Error Rate of Each Estimator values by using 4-Fold Cross Validation

In the code, uncomment and execute the *AdaBoost_cross_validation ()* function to get the result. It took about 9 minutes to get the results.

Cross Validation Results of AdaBoost

- Fold #1: Best Estimator value is 24 with Error rate of 49.61%
- Fold #2: Best Estimator value is 4 with Error rate of 49.30%
- Fold #3: Best Estimator value is 45 with Error rate of 47.10%
- Fold #4: Best Estimator value is 5 with Error rate of 42.98%
- Average Error Rate is 47.25%

Receiver Operating Characteristic (ROC) Curve of AdaBoost

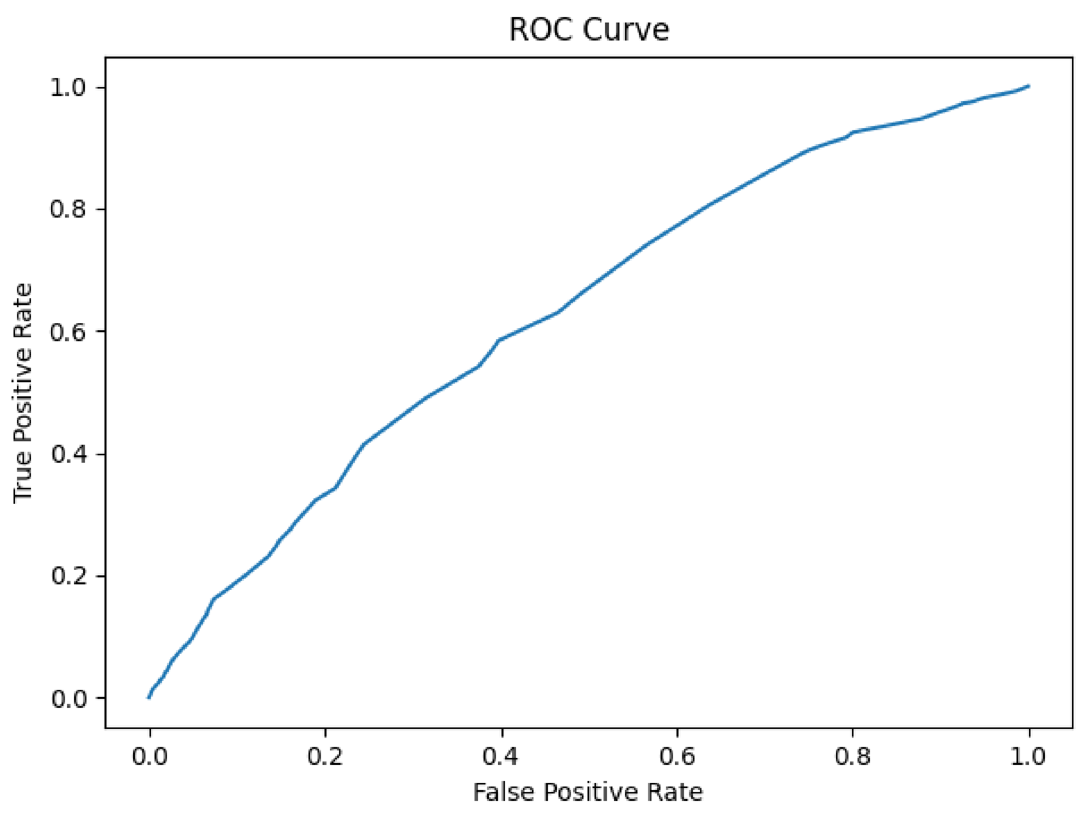

Figure GNB 1 – ROC Curve of the AdaBoost Algorithm

Through the Receiver Operating Characteristic (ROC) calculation provided by Sklearn, we got the above graph. The positive class we set is 2. From this graph, we can see that the overall curve slightly tends to be positive, and the result is quite bad.

Support Vector Machines (SVM)

Support Vector Machines (SVM) is a popular machine learning algorithm used for classification and regression tasks. It is a type of supervised learning algorithm that works by constructing a hyperplane or a set of hyperplanes in a high-dimensional space that can be used for classification or regression tasks.

In SVM, we aim to find the hyperplane that maximally separates the two classes, which are represented by the training data points. The hyperplane is constructed by finding the support vectors, which are the training data points that are closest to the hyperplane. The distance between the support vectors and the hyperplane is called the margin, and SVMs seek to maximize this margin.

Computational Times of SVM

From the results of the code, we recorded the computation time for the Support Vector Machines algorithm with RBF kernel by using Sklearn as follows:

- Computational Times for Training data is 1240015.46 milliseconds.
- Computational Times for Testing data is 326333.92 milliseconds.

Confusion Matrix of SVM

Based on the results generated from the testing dataset using the Support Vector Machines algorithm with RBF kernel, we can plot the confusion matrix as follows: Overall, Error rate is 51.42%

Class B misclassified as Class A (**False positive**) **8843 (69.36%)**	Class A correctly classified (**True positive**) **8480 (66.51%)**
Class B correctly classified (**True negative**) **3907 (30.64%)**	Class A misclassified as Class B (**False negative**) **4270 (33.49%)**

Cross Validation of SVM

Cross-validation is applied to evaluate the performance of using SVM on the accelerometer dataset. We have used the 4-fold cross validation method from the training set and validation set. A report is generated called "SVM Cross Validation Partition Data.xlsx".

In the code, uncomment and execute the *SVM_cross_validation ()* function to get the result. It took about few seconds to get the results.

Cross Validation Results of SVM

- Fold #1: Error rate is 50.00%
- Fold #2: Error rate is 50.01%
- Fold #3: Error rate is 48.69%
- Fold #4: Error rate is 49.73%
- Average Error Rate is 49.61%

Receiver Operating Characteristic (ROC) Curve of SVM

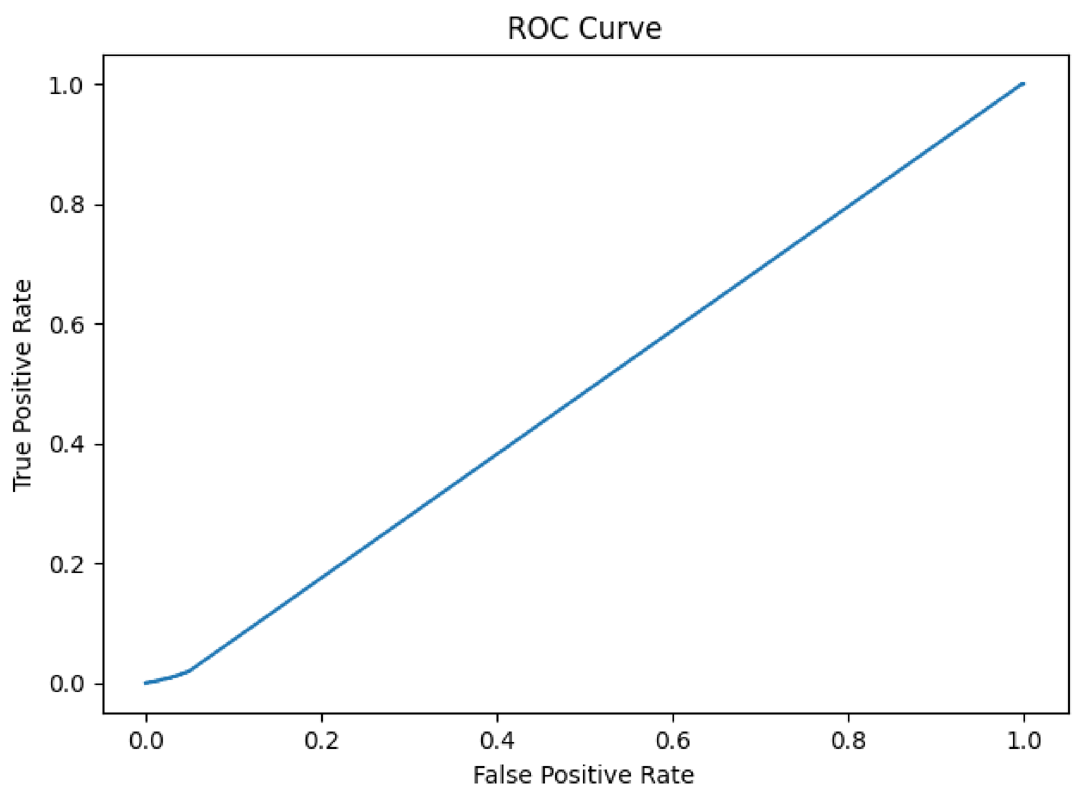

Figure SVM 1 – ROC Curve of the SVM Algorithm

Through the Receiver Operating Characteristic (ROC) calculation provided by Sklearn, we got the above graph. The positive class we set is 2. From this graph, we can see that the overall curve tends to be negative, and the result is very bad.

Comparison of the Classifiers

	K-Nearest Neighbours	Gaussian Naive Bayes	Adaptive Boosting	Support Vector Machines
Computational times for training	76.81 milliseconds	11.99 milliseconds	385.99 milliseconds	1240015.46 milliseconds
Computational times for testing	2238.01 milliseconds	6.01 milliseconds	59.87 milliseconds	326333.92 milliseconds
Cross Validation Average Error Rate	37.38%	48.76%	47.25%	49.61%
ROC Curve	It's OK	It's OK	Quite Bad	Very Bad

From the summary table above, we can briefly see the pro and cons of each classifier. The data set has lots of overlapping data points from two classes, therefore, it is a little bit hard to classify the data set we have. From all the results, we can see the best average error rate is only 37.38% from K-Nearest Neighbours classification, and the worst average error rate is 49.61% from Support Vector Machines classification.

In terms of computational times for training and testing data, Gaussian Naïve Bayes classification has the best score under 12 milliseconds. Support Vector Machines classification has the worst scores, the computational time is even exceeded 20 minutes.

In terms of ROC Curve, there is no classifier can do a good job on it. The worst result once again comes to the Support Vector Machines classification.

Conclusion

KNN is a non-parametric method, meaning it does not make any assumptions about the underlying distribution of the data, which makes it more flexible and robust in handling different types of data. The training process of KNN is fast since it only stores the training data and makes predictions based on the nearest neighbors. Therefore, KNN can perform well when there is a large amount of training data available and when the decision boundaries are complex. The classifier yields a best result in accuracy and great result in computational time.

AdaBoost is not prone to overfitting, as it uses a weighted sample selection process that focuses on the misclassified samples, which can improve generalization. It can handle non-linear decision boundaries by using a variety of base classifiers. Because the elements of different classes on the data are almost overlapping each other, there is no noisy data and outliers problem. This is the reason that AdaBoost performs quite well compared to other classifiers on the data set. The classifier yields a good result in accuracy and great result in computational time.

GNB is computationally efficient and can make predictions quickly, which is useful in real-time applications. It requires a small amount of training data compared to other algorithms, which can be beneficial when dealing with limited data. But the assumption of conditional independence may not hold in some cases, which can lead to poor performance. The classifier yields an average result in accuracy and best result in computational time.

SVMs can be computationally expensive and time-consuming, especially when dealing with large datasets or non-linear problems. It requires careful selection of hyperparameters, such as the kernel function, regularization parameter, and kernel bandwidth, to achieve optimal performance. The classifier yields a worst result in accuracy and worst result in computational time.

Overall, the best fitting to worst fitting classifiers on out data set is as follows: KNN > AdaBoost > GNB > SVMs.