GE 461 Introduction to Data Science

Project 5 Report

Oğuz Altan – 21600966

This project is about data stream mining. Classifying data streams is a challenging problem due to time and memory limitations as well as variation in data distribution. The aim is to effectively classify the data as they continuously keep entering to the system.

<u>Work Done</u>

1) Dataset Generation

a. RBF Dataset

In this part, we generate a dataset with 10,000 instances using "Random RBF Generator" and write it into a file called "RBF Dataset". This dataset has 10 features and 2 class labels. As an instance, first two samples of this dataset is given below. Rightmost column shows the corresponding class labels.

0.4495	1.0920	0.3477	0.9218	0.1950	0.2883	0.8293	0.2684	0.8096	0.2385	1
0.2034	0.5613	0.7665	0.6108	0.6045	0.8873	0.0424	0.0914	0.7037	0.6575	1

Table 1: First Two Rows of RBF Dataset

b. RBF Dataset10

In this part, we generate a dataset with 10,000 instances using "Random RBF Generator Drift" with drift speed of 10 and write it into a file called "RBF Dataset 10". This dataset has 10 features and 2 class labels. As an instance, first two samples of this dataset is given below. Rightmost column shows the corresponding class labels.

1.0698	0.9649	0.9474	1.0163	1.1074	1.0386	1.071	1.114	1.102	0.8966	1
0.4454	0.0856	0.3749	0.0582	0.7484	0.8873	0.0331	0.0635	0.0390	0.0786	0

Table 2: First Two Rows of RBF Dataset 10

c. RBF Dataset70

In this part, we generate a dataset with 10,000 instances using "Random RBF Generator Drift" with drift speed of 70 and write it into a file called "RBF Dataset 70". This dataset has 10 features and 2 class labels. As an instance, first two samples of this dataset is given below. Rightmost column shows the corresponding class labels.

1.1282	1.1156	1.0490	1.0580	0.7652	1.0434	0.8714	0.87682	1.0898	1.0725	1
0.0820	0.0120	0.0373	0.0778	0.0349	0.0218	0.0014	0.0048	0.0858	0.01430	1

Table 3: First Two Rows of RBF Dataset 70

2) Data Stream Classification with Three Separate Online Single Classifiers: HT, NB, MLP

In this section, a script in Python that constructs and trains the following online classifiers using three RBF Datasets generated in the section A:

a. Hoeffding Tree as HT online learner
b. Naïve Bayes as NB online learner
c. Multilayer Perceptron as MLP composed of 4 hidden layers of 200 neurons

These online learners are implemented scikit-multiflow and scikit- learn packages. The evaluations are made using partial_fit methods, this method uses incremental learning. In online learning, we use the data one by one or in a sliding window. Online learning uses Interleaved-Test-Then-Train approach. This approach proposes the systems where the incoming data chunk is first tested using the model, and after it is used to train the model, as the name of this approach implies. As it can be seen on the code, the data chunks, one by one or as a sliding windows, is firs tested and then trained on the model, using a for loop that iterates over the corresponding dataset. The accuracy results for these online single classifiers are given below in a table:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Hoeffding Tree online learner	75.99%	55.13%	53.60%
Naïve Bayes online learner	67.80%	48.74%	50.17%
Multilayer Perceptron online learner	67.73%	50.57%	50.11%

Table 4: Accuracy Results for Thee Datasets for Online Learners HT, NB and MLP

The accuracy results, as decimal values, are also given below as text output:

- Accuracy of Hoeffding Tree with Online Classification using RBF Dataset: 0.75997599759976
- Accuracy of Hoeffding Tree with Online Classification using RBF Dataset 10: 0.5513551355135513
- Accuracy of Hoeffding Tree with Online Classification using RBF Dataset 70: 0.536053605360536
- Accuracy of Naive Bayes with Online Classification using RBF Dataset: 0.678067806780678
- Accuracy of Naive Bayes with Online Classification using RBF Dataset 10: 0.48744874487448747
- Accuracy of Naive Bayes with Online Classification using RBF Dataset 70: 0.5017501750175017
- Accuracy of Multilayer Perceptron with Online Classification using RBF Dataset: 0.6773677367736773
- Accuracy of Multilayer Perceptron with Online Classification using RBF Dataset 10: 0.5057505750575058
- Accuracy of Multilayer Perceptron with Online Classification using RBF Dataset 70: 0.5011501150115012

3) Data Stream Classification with Two Online Ensemble Classifiers: MV, WMV

In this section, a script in Python that constructs and trains the following ensemble classifiers that combines HT, NB, and MLP using three RBF Datasets generated in the section A:

a. Majority voting rule as MV
b. Weighted majority voting rule as WMV

Discussing ensemble learners, these learners combine single classifiers to obtain a strengthen accuracy power and higher prediction rate. The ensemble classifiers used in this section are majority voting rule ensemble classifier and weighted majority voting rule ensemble classifier. Majority voting rule is based on choosing the predicted label according to the majority of classifier's prediction. For instance, out of three classifiers, if two of them predict a sample's label as "0" and the other does it as "1", then the ensemble classifier chooses sample's label as prediction of "0". This system increases the accuracy rate and power of the classification process. Similarly, for the weighted majority-voting rule, each classifier has different importance of voting, this time, all of the classifiers does not contribute to the voting equally. Again, Interleaved-Test-Then-Train approach is used. As it can be seen on the code, the data chunks, one by one or as a sliding windows, is firs tested and then trained on the model, using a for loop that iterates over the corresponding dataset. The predictions of all of classifiers are combined using majority and weighted majority voting ensemble system. The accuracy results for these online ensemble classifiers are given below in a table:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Majority voting rule online ensemble classifier	71.52%	71.25%	70.22%
Weighted majority voting rule online ensemble classifier	81.71%	79.97%	77.79%

Table 5: Accuracy Results for Thee Datasets for Online Ensemble Classifiers MV and WMV

The accuracy results, as decimal values, are also given below as text output:

- Accuracy of Majority Voting Online Ensemble Classification using RBF Dataset: 0.7152525252525253
- Accuracy of Majority Voting Online Ensemble Classification using RBF Dataset 10: 0.7125712571257126
- Accuracy of Majority Voting Online Ensemble Classification using RBF Dataset 70: 0.7022430496850381
- Accuracy of Weighted Majority Voting Online Ensemble Classification using RBF Dataset: 0.8171171171171171
- Accuracy of Weighted Majority Voting Online Ensemble Classification using RBF Dataset 10: 0.7997997997997998
- Accuracy of Weighted Majority Voting Online Ensemble Classification using RBF Dataset 70: 0.777977977977978

4) Batch Classification with Three Separate Batch Single Classifiers: HT, NB, MLP

In this section, a script in Python that constructs and trains the following batch classifiers using three RBF Datasets generated in the section A:

a. Hoeffding Tree as HT batch learner
b. Naïve Bayes as NB batch learner
c. Multilayer Perceptron as MLP composed of 4 hidden layers of 200 neurons

These batch learners are implemented using scikit-multiflow and scikit- learn packages. This method uses batch learning. Opposite of online learning, batch learning uses the whole dataset to train the model. This method is based on traditional learning methods. First, the dataset is split into training and test datasets. Then, the model is trained using the whole training dataset and then test dataset is used to measure the accuracy rate and prediction of the power. Analyzing the work done in this project, the datasets are split into training and test datasets, as it is mentioned, then the batch classifier models are trained using fit() method in the scikit-multiflow and scikit- learn packages. Then, the labels for test datasets are predicted using predict() method. The accuracy is finally calculated and printed. The accuracy results for these batch single classifiers are given below in a table:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Hoeffding Tree batch learner	77.82%	58.94%	51.20%
Naïve Bayes batch learner	69.38%	52.60%	50.12%
Multilayer Perceptron batch learner	84.90%	75.54%	56.00%

Table 6: Accuracy Results for Thee Datasets for Batch Learners HT, NB and MLP

The accuracy results, as decimal values, are also given below as text output:

- Accuracy score for Hoeffding Tree Batch Classification for RBF Dataset: 0.7782
- Accuracy score for Hoeffding Tree Batch Classification for RBF Dataset 10: 0.5894
- Accuracy score for Hoeffding Tree Batch Classification for RBF Dataset 70: 0.5120
- Accuracy score for Naive Bayes Batch Classification for RBF Dataset: 0.6938
- Accuracy score for Naive Bayes Batch Classification for RBF Dataset 10: 0.5260
- Accuracy score for Naive Bayes Batch Classification for RBF Dataset 70: 0.5012
- Accuracy score for Multilayer Perceptron with Batch Classification for RBF Dataset: 0.8490
- Accuracy score for Multilayer Perceptron with Batch Classification for RBF Dataset 10: 0.7554
- Accuracy score for Multilayer Perceptron with Batch Classification for RBF Dataset 70: 0.5600

5) Batch Classification with Two Batch Ensemble Classifiers: MV, WMV

In this section, a script in Python that constructs and trains the following ensemble classifiers that combines HT, NB, and MLP using three RBF Datasets generated in the section A:

a. Majority voting rule as MV
b. Weighted majority voting rule as WMV

Discussing ensemble learners, these learners combine single classifiers to obtain a strengthen accuracy power and higher prediction rate. The ensemble classifiers used in this section are majority voting rule ensemble classifier and weighted majority voting rule ensemble classifier. Majority voting rule is based on choosing the predicted label according to the majority of classifier's prediction. For instance, out of three classifiers, if two of them predict a sample's label as "0" and the other does it as "1", then the ensemble classifier chooses sample's label as prediction of "0". This system increases the accuracy rate and power of the classification process. Similarly, for the weighted majority-voting rule, each classifier has different importance of voting, this time, all of the classifiers does not contribute to the voting equally. Instead of Interleaved-Test-Then-Train approach, a straightforward train and test approach is used, as batch classification premises.

The predictions of all of classifiers are combined using majority and weighted majority voting ensemble system. The accuracy results for these batch ensemble classifiers are given below in a table:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Majority voting rule batch ensemble classifier	83.19%	50.31%	52.17%
Weighted majority voting rule batch ensemble classifier	50.56%	50.52%	50.54%

Table 7: Accuracy Results for Thee Datasets for Batch Ensemble Classifiers MV and WMV

The accuracy results, as decimal values, are also given below as text output:

- Accuracy of Majority Voting Batch Ensemble Classification using RBF Dataset: 0.8319663932786557
- Accuracy of Majority Voting Batch Ensemble Classification using RBF Dataset 10: 0.5031006201240248
- Accuracy of Majority Voting Batch Ensemble Classification using RBF Dataset 70: 0.5217043408681736
- Accuracy of Weighted Majority Voting Batch Ensemble Classification using RBF Dataset: 0.5056
- Accuracy of Weighted Majority Voting Batch Ensemble Classification using RBF Dataset 10: 0.5052
- Accuracy of Weighted Majority Voting Batch Ensemble Classification using RBF Dataset 70: 0.5054

6) Comparison of Models

a. In this part, we compare temporal accuracies of online classifiers, the ones generated in the steps 2, 3, using Interleaved-Test-Then-Train approach, using instances of the datasets generated in first step. For sake of easiness, the accuracies of three online learners are given below:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Hoeffding Tree online learner	75.99%	55.13%	53.60%
Naïve Bayes online learner	67.80%	48.74%	50.17%
Multilayer Perceptron online learner	67.73%	50.57%	50.11%

Table 8: Accuracy Results for Thee Datasets for Online Learners HT, NB and MLP

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Majority voting rule online ensemble classifier	71.52%	71.25%	70.22%
Weighted majority voting rule online ensemble classifier	81.71%	79.97%	77.79%

Table 9: Accuracy Results for Thee Datasets for Online Ensemble Classifiers MV and WMV

Comparing online single classifiers, it can be seen that accuracies are similar for each of datasets. Hoeffding Tree online learner seems to have higher accuracy then others but the difference is not very significant. Analyzing online ensemble classifiers, it can be seen that accuracies for Majority voting rule and Weighted Majority voting rule ensemble classifiers have higher accuracies than online single classifiers. In addition, the accuracies of ensemble classifiers do not change significantly concerning the different datasets.

b. In the comparison of online classifiers, different window sizes are tried. The reasoning behind the windows size is that, in online learning, instead of giving the data one by one as input, the data chunks are given within a window. Window size is the size of this data chunk given as input to the system. The results for different window size are given below:

Results for windows size of 1, i.e. data is given one by one:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Hoeffding Tree online learner	75.99%	55.13%	53.60%
Naïve Bayes online learner	67.80%	48.74%	50.17%
Multilayer Perceptron online learner	67.73%	50.57%	50.11%

Table 10: Accuracy Results for Thee Datasets for Online Learners HT, NB and MLP for Window Size = 1

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Majority voting rule online ensemble classifier	71.52%	71.25%	70.22%
Weighted majority voting rule online ensemble classifier	81.71%	79.97%	77.79%

Table 11: Accuracy Results for Thee Datasets for Online Ensemble Classifiers MV and WMV

Results for windows size of 20, i.e. 20 samples of dataset is given to the system as one input:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Hoeffding Tree online learner	78.15%	59.37%	57.36%
Naïve Bayes online learner	69.60%	49.21%	51.91%
Multilayer Perceptron online learner	66.44%	49.94%	51.41%

Table 12: Accuracy Results for Thee Datasets for Online Learners HT, NB and MLP for Window Size = 20

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Majority voting rule online ensemble classifier	73.92%	72.44%	72.39%
Weighted majority voting rule online ensemble classifier	81.90%	80.26%	78.14%

Table 13: Accuracy Results for Thee Datasets for Online Ensemble Classifiers MV and WMV for Window Size = 20

Results for windows size of 200, i.e. 200 samples of dataset is given to the system as one input:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Hoeffding Tree online learner	79.69%	60.48%	59.29%
Naïve Bayes online learner	72.40%	50.74%	52.46%
Multilayer Perceptron online learner	68.19%	51.50%	52.03%

Table 14: Accuracy Results for Thee Datasets for Online Learners HT, NB and MLP for Window Size = 200

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Majority voting rule online ensemble classifier	76.42%	74.50%	73.61%
Weighted majority voting rule online ensemble classifier	83.52%	82.68%	80.93%

Table 15: Accuracy Results for Thee Datasets for Online Ensemble Classifiers MV and WMV for Window Size = 200

From the tables, it can be seen that the accuracies increase as the windows size increases. This situation leads to the result that windows size affects the result of the predictions. Therefore, window sizes are influential in understanding the performance of the methods.

c. Comparing ensemble methods and individual models, the reasoning behind ensemble methods should be discussed. As mentioned in sections 3 and 5, the ensemble learners combine single classifiers to obtain a strengthen accuracy power and higher prediction rate. The ensemble classifiers that are used in this project are majority voting rule ensemble classifier and weighted majority voting rule ensemble classifier. Majority voting rule is based on choosing the predicted label according to the majority of classifier's prediction. For instance, out of three classifiers, if two of them predict a sample's label as "0" and the other does it as "1", then the ensemble classifier chooses sample's label as prediction of "0". This system increases the accuracy rate and power of the classification process. Similarly, for the weighted majority-voting rule, each classifier has different importance of voting, this time, all of the classifiers does not contribute to the voting equally. In fact, the weights show the importance of the individual classifiers in voting process. As a result, ensemble learners combines accuracy and prediction powers of individual classifiers, therefore they are expected to have a higher performance and accuracy rate compared to individual classifiers. These ensemble classifiers can be used for online learning to increase the overall accuracy, as well as for batch learning. To show that, the results of individual and ensemble classifiers are given for the sake of easiness:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Hoeffding Tree online learner	75.99%	55.13%	53.60%
Naïve Bayes online learner	67.80%	48.74%	50.17%
Multilayer Perceptron online learner	67.73%	50.57%	50.11%

Table 16: Accuracy Results for Thee Datasets for Online Learners HT, NB and MLP

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Majority voting rule online ensemble classifier	71.52%	71.25%	70.22%
Weighted majority voting rule online ensemble classifier	81.71%	79.97%	77.79%

Table 17: Accuracy Results for Thee Datasets for Online Ensemble Classifiers MV and WMV

It can be seen from the tables 16 and 17 that the accuracies of online ensemble classifiers are higher than the accuracies of online single classifiers

d. Comparing all online and batch models (single and ensemble), first thing to notice is that ensemble classifiers have higher overall accuracies than the individual single classifiers, as mentioned in the previous subsection, subsection c). Comparing the online and batch classifiers, for the sake of easiness, the results for online and batch single classifiers are given below:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Hoeffding Tree online learner	75.99%	55.13%	53.60%
Naïve Bayes online learner	67.80%	48.74%	50.17%
Multilayer Perceptron online learner	67.73%	50.57%	50.11%

Table 18: Accuracy Results for Thee Datasets for Online Learners HT, NB and MLP

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Hoeffding Tree batch learner	77.82%	58.94%	51.20%
Naïve Bayes batch learner	69.38%	52.60%	50.12%
Multilayer Perceptron batch learner	84.90%	75.54%	56.00%

Table 19: Accuracy Results for Thee Datasets for Online Learners HT, NB and MLP

Comparing these results, it can be seen that overall accuracies for batch classifiers are higher than online classifiers. The reason lies behind the aim of using online and batch learning. Online learning is generally used to work with very big datasets, for instance data in internet, videos uploaded to Youtube, Tweets, Facebook posts, Instagram photos, Google searches and so on. There is a huge flow of data stream in nowadays. To be able to analyze this big data, batch learning is inappropriate and inefficient in terms of memory space used for processing and very time consuming, because batch learning uses the whole dataset for train and testing. However, considering datastreams, data flows with a very high speed, therefore there is not any large margin for time and memory usage. Therefore, time and memory managements should be appropriate to process this big data, datastreams. Online learning, however, uses a different technique than batch learning. Online learning takes the data one by one or in a window, as demonstrated on subsection b). The reasoning behind online learning is that, the system sees the data package once and uses Interleaved-Test-Then-Train approach, meaning that it first tests the model with that incoming data chunk and then trains the model. Thus, processing speed increases significantly and the time used for learning is much shorter. In addition, as the dataset are set together and saved as in batch learning, batch learning needs more memory space, considering online learning, the system sees that data chunk once and uses it, then discards it. For the big data processing, memory space is very important factor and online learning overcomes it using the aforementioned methods.

As a result, for online learning, time and memory issues are more important, therefore, it is not high accuracy oriented, unlike the batch learning. Due to these facts, online learning performs worse than batch learning considering the overall accuracy.

e. An improvement for the prediction accuracy of online classifiers is implemented. This improvement is about weighted majority voting ensemble classifiers. As weights, the algorithm takes the individual accuracy performances of the corresponding single online classifiers and normalize the total of the accuracy scores for each of the single online classifiers to 1, i.e. the weights of each online single classifiers that are combined in weighted majority voting ensemble online classifier add up to 1. This improvement is tested and the accuracies are already given in the table in the section 3. This table is given below again, for the sake of easiness:

Classifier/Dataset Accuracies	RBF Dataset	RBF Dataset10	RBF Dataset70
Majority voting rule online ensemble classifier	71.52%	71.25%	70.22%
Weighted majority voting rule online ensemble classifier	81.71%	79.97%	77.79%

Table 20: Accuracy Results for Thee Datasets for Online Ensemble Classifiers MV and WMV

f. The time measurements in seconds, i.e how long the process lasted for the classifiers are given below:

Classifier/Time	Time (in second)
Hoeffding Tree online learner	46.4667
Naïve Bayes online learner	41.6795
Multilayer Perceptron online learner	244.1010

Table 21: Time Measurements for Three Classifiers of Training and Test of All Datasets, Summed Up

These time measurements are for the process of time of training and test of all datasets, summed up. In terms of efficiency, for instance time usage, it can be seen that Multilayer Perceptron classifier uses more time to perform prediction compared to Naïve Bayes and Hoeffding Tree.

Appendix

A) Source Code

project5.py

```
# -*- coding: utf-8 -*-

"""

@author: Oguz Altan

@Date: 24.05.20

@Title: GE461 Stream Mining Project

"""


import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from skmultiflow.data.random_rbf_generator import RandomRBFGenerator

from skmultiflow.data.random_rbf_generator_drift import RandomRBFGeneratorDrift

from skmultiflow.trees import HoeffdingTree

from skmultiflow.bayes import NaiveBayes

from sklearn.neural_network import MLPClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

from timeit import default_timer as timer


#Creating 10,000 instances using Random RBF Generator

stream = RandomRBFGenerator(model_random_state=99, sample_random_state=50, n_classes=2,
n_features = 10, n_centroids=50)

stream.prepare_for_use()


features_tuple,class_tuple = stream.next_sample(10000)

RBF_Dataset = pd.DataFrame(np.hstack((features_tuple,np.array([class_tuple]).T)))

RBF_Dataset.to_csv('RBF Dataset.csv')
```

```
#Creating 10,000 instances using Random RBF Generator Drift with Drift Speed  = 10

stream10 = RandomRBFGeneratorDrift(model_random_state=99, sample_random_state = 50,
n_classes = 2, n_features = 10, n_centroids = 50, change_speed = 10, num_drift_centroids=50)

stream10.prepare_for_use()


features_tuple10,class_tuple10 = stream10.next_sample(10000)

RBF_Dataset10 = pd.DataFrame(np.hstack((features_tuple10,np.array([class_tuple10]).T)))

RBF_Dataset10.to_csv('RBF Dataset 10.csv')


features_tuple10,class_tuple10 = stream10.next_sample(10000)

RBF_Dataset10 = pd.DataFrame(np.hstack((features_tuple10,np.array([class_tuple10]).T)))

RBF_Dataset10.to_csv('RBF Dataset 10.csv')


#Creating 10,000 instances using Random RBF Generator Drift with Drift Speed  = 70

stream70 = RandomRBFGeneratorDrift(model_random_state=99, sample_random_state = 50,
n_classes = 2, n_features = 10, n_centroids = 50, change_speed = 70, num_drift_centroids=50)

stream70.prepare_for_use()


features_tuple70,class_tuple70 = stream70.next_sample(10000)

RBF_Dataset70 = pd.DataFrame(np.hstack((features_tuple70,np.array([class_tuple70]).T)))

RBF_Dataset70.to_csv('RBF Dataset 70.csv')


train_features, test_features, train_labels, test_labels = train_test_split(RBF_Dataset.iloc[:, 0:10],

                                              RBF_Dataset.iloc[:, 10],

                                              test_size=0.5, random_state=42)

train_features10, test_features10, train_labels10, test_labels10 =
train_test_split(RBF_Dataset10.iloc[:, 0:10],

                                              RBF_Dataset10.iloc[:, 10],

                                              test_size=0.5, random_state=42)

train_features70, test_features70, train_labels70, test_labels70 =
train_test_split(RBF_Dataset70.iloc[:, 0:10],
                                          RBF_Dataset70.iloc[:, 10],
                                          test_size=0.5, random_state=42)

#%%Hoeffding Tree with Online Classification
HT = HoeffdingTree()

start = timer()

correctness_dist = []
for i in range(10000):
  x, y = stream.next_sample()
  predictHT = HT.predict(x)

  if y == predictHT:
    correctness_dist.append(1)
  else:
    correctness_dist.append(0)

  HT.partial_fit(x, y.ravel())

time = [i for i in range(1, 10000)]
accuracy = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]
plt.plot(time, accuracy)
print("Accuracy of Hoeffding Tree with Online Classification using RBF Dataset: ",accuracy[-1])

correctness_dist = []
for i in range(10000):
   x, y = stream10.next_sample()
   predictHT10 = HT.predict(x)
```

```
    if y == predictHT10:
      correctness_dist.append(1)
    else:
      correctness_dist.append(0)

    HT.partial_fit(x, y)

time = [i for i in range(1, 10000)]
accuracy10 = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]
plt.plot(time, accuracy10)
print("Accuracy of Hoeffding Tree with Online Classification using RBF Dataset 10: ",accuracy10[-1])

correctness_dist = []
for i in range(10000):
    x, y = stream70.next_sample()
    predictHT70 = HT.predict(x)

    if y == predictHT70:
      correctness_dist.append(1)
    else:
      correctness_dist.append(0)

    HT.partial_fit(x, y)

time = [i for i in range(1, 10000)]
accuracy70 = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]
plt.plot(time, accuracy70)
print("Accuracy of Hoeffding Tree with Online Classification using RBF Dataset 70: ",accuracy70[-1])
end = timer()
print(end - start)
```

```
#%%Naive Bayes with Online Classification

NB = NaiveBayes()

start = timer()


correctness_dist = []

for i in range(10000):

  x, y = stream.next_sample()

  predictNB = NB.predict(x)


  if y == predictNB:

    correctness_dist.append(1)

  else:

    correctness_dist.append(0)


  NB.partial_fit(x, y)


time = [i for i in range(1, 10000)]

accuracy = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]

plt.plot(time, accuracy)

print("Accuracy of Naive Bayes with Online Classification using RBF Dataset: ",accuracy[-1])


correctness_dist = []

for i in range(10000):

   x, y = stream10.next_sample()

   predictNB10 = NB.predict(x)


   if y == predictNB10:

     correctness_dist.append(1)

   else:
      correctness_dist.append(0)

    NB.partial_fit(x, y)

time = [i for i in range(1, 10000)]
accuracy = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]
plt.plot(time, accuracy)
print("Accuracy of Naive Bayes with Online Classification using RBF Dataset 10: ",accuracy[-1])

correctness_dist = []
for i in range(10000):
    x, y = stream70.next_sample()
    predictNB70 = NB.predict(x)

    if y == predictNB70:
      correctness_dist.append(1)
    else:
      correctness_dist.append(0)

    NB.partial_fit(x, y)

time = [i for i in range(1, 10000)]
accuracy = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]
plt.plot(time, accuracy)
print("Accuracy of Naive Bayes with Online Classification using RBF Dataset 70: ",accuracy[-1])
end = timer()
print(end - start)

#%%Multilayer Perceptron with Online Classification
MLP = MLPClassifier(hidden_layer_sizes = [200,200,200,200],
                max_iter = 10000,
                activation = 'tanh',
                batch_size= 1,
                solver = 'adam',
                random_state = 42)
start = timer()
correctness_dist = []

for i in range(10000):

  MLP.partial_fit(x, y,classes = [0,1])

  x, y = stream.next_sample()
  predictMLP = MLP.predict(x)

  if y == predictMLP:
    correctness_dist.append(1)
  else:
    correctness_dist.append(0)

 # MLP.partial_fit(x, y)

time = [i for i in range(1, 10000)]
accuracy = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]
plt.plot(time, accuracy)
print("Accuracy of Multilayer Perceptron with Online Classification using RBF Dataset: ",accuracy[-1])

correctness_dist = []

for i in range(10000):
   MLP.partial_fit(x, y,classes = [0,1])


   x, y = stream10.next_sample()

   predictMLP10 = MLP.predict(x)


   if y == predictMLP10:

     correctness_dist.append(1)

   else:

     correctness_dist.append(0)


  # MLP.partial_fit(x, y)


time = [i for i in range(1, 10000)]

accuracy = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]

plt.plot(time, accuracy)

print("Accuracy of Multilayer Perceptron with Online Classification using RBF Dataset 10: ",accuracy[-
1])


correctness_dist = []


for i in range(10000):


   MLP.partial_fit(x, y,classes = [0,1])


   x, y = stream70.next_sample()

   predictMLP70 = MLP.predict(x)


   if y == predictMLP70:

     correctness_dist.append(1)

   else:
```

```
            correctness_dist.append(0)


    # MLP.partial_fit(x, y)


time = [i for i in range(1, 10000)]
accuracy = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]
plt.plot(time, accuracy)
print("Accuracy of Multilayer Perceptron with Online Classification using RBF Dataset 70: ",accuracy[-
1])
end = timer()
print(end - start)
#%%Ensemble Learning with Majority Voting for Online Classification
correctness_dist = []


for i in range(10000):
    if i < 100:
        X, y = stream.next_sample()
        HT.partial_fit(X, y, classes=[0, 1])
        NB.partial_fit(X, y, classes=[0, 1])
        MLP.partial_fit(X, y, classes=[0, 1])
    else:
        X, y = stream.next_sample()
        prediction = []
        prediction.append(HT.predict(X))
        prediction.append(NB.predict(X))
        prediction.append(MLP.predict(X))


        if ((sum(prediction)/2) <1):
            pred = 0
        else:
            pred = 1
        if pred == y:
            correctness_dist.append(1)
        else:
            correctness_dist.append(0)

        HT.partial_fit(X, y)
        NB.partial_fit(X, y)
        MLP.partial_fit(X, y)

time = [i for i in range(1, 10000)]
accuracy = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]
plt.plot(time, accuracy)
print("Accuracy of Majority Voting Online Ensemble Classification using RBF Dataset: ",accuracy[-1])

for i in range(10000):
    if i < 100:
        X, y = stream10.next_sample()
        HT.partial_fit(X, y, classes=[0, 1])
        NB.partial_fit(X, y, classes=[0, 1])
        MLP.partial_fit(X, y, classes=[0, 1])
    else:
        X, y = stream10.next_sample()
        prediction = []
        prediction.append(HT.predict(X))
        prediction.append(NB.predict(X))
        prediction.append(MLP.predict(X))

        if ((sum(prediction)/2) <1):
            pred = 0
        else:
            pred = 1

        if pred == y:
            correctness_dist.append(1)
        else:
            correctness_dist.append(0)

        HT.partial_fit(X, y)
        NB.partial_fit(X, y)
        MLP.partial_fit(X, y)

time = [i for i in range(1, 10000)]
accuracy = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]
plt.plot(time, accuracy)
print("Accuracy of Majority Voting Online Ensemble Classification using RBF Dataset 10: ",accuracy[-
1])

for i in range(10000):
    if i < 100:
        X, y = stream70.next_sample()
        HT.partial_fit(X, y, classes=[0, 1])
        NB.partial_fit(X, y, classes=[0, 1])
        MLP.partial_fit(X, y, classes=[0, 1])
    else:
        X, y = stream70.next_sample()
        prediction = []
        prediction.append(HT.predict(X))
        prediction.append(NB.predict(X))
        prediction.append(MLP.predict(X))
        if ((sum(prediction)/2) <1):
            pred = 0
        else:
            pred = 1

        if pred == y:
            correctness_dist.append(1)
        else:
            correctness_dist.append(0)

        HT.partial_fit(X, y)
        NB.partial_fit(X, y)
        MLP.partial_fit(X, y)

time = [i for i in range(1, 10000)]
accuracy = [sum(correctness_dist[:i])/len(correctness_dist[:i]) for i in range(1, 10000)]
plt.plot(time, accuracy)
print("Accuracy of Majority Voting Online Ensemble Classification using RBF Dataset 70: ",accuracy[-
1])

#Ensemble Learning with Weighted Majority Voting for Batch Classification
pred_weighted = []
yvec = []

HT_W = 0.2
NB_W = 0.3
MLP_W = 0.5

# totac = accuracy_score(test_labels, predHT) + accuracy_score(y_test, predNB) +
accuracy_score(y_test, predMLP)
```

```
# HT_W = accuracy_score(test_labels, predHT)*1/totacc

# NB_W = accuracy_score(test_labels, predNB)*1/totacc

# MLP_W = accuracy_score(test_labels, predMLP)*1/totacc


for i in range(10000):

   if i < 10:

      X, y = stream.next_sample()

      HT.partial_fit(X, y, classes=[0, 1])

      NB.partial_fit(X, y, classes=[0, 1])

      MLP.partial_fit(X, y, classes=[0, 1])

   else:

      X, y = stream.next_sample()


      yvec.append(y)


      prob_0 = (HT_W * HT.predict_proba(X)[0][0]) + (NB_W * NB.predict_proba(X)[0][0]) + (MLP_W *
MLP.predict_proba(X)[0][0])

      prob_1 = (HT_W * HT.predict_proba(X)[0][1]) + (NB_W * NB.predict_proba(X)[0][1]) + (MLP_W *
MLP.predict_proba(X)[0][1])


      if prob_1 > prob_0:

         pred_weighted.append(1)

      else:

         pred_weighted.append(0)


      HT.partial_fit(X, y)

      NB.partial_fit(X, y)

      MLP.partial_fit(X, y)


accurate = accuracy_score(yvec,pred_weighted)

print("Accuracy of Weighted Majority Voting Online Ensemble Classification using RBF Dataset:
",accurate)
```

```
pred_weighted10 = []

yvec10 = []


HT_W10 = 0.2

NB_W10 = 0.3

MLP_W10 = 0.5


# totacc10 = accuracy_score(test_labels10, predHT10) + accuracy_score(y_test10, predNB10) +
accuracy_score(y_test10, predMLP10)


# HT_W10 = accuracy_score(test_labels10, predHT10)*1/totacc10

# NB_W10 = accuracy_score(test_labels10, predNB10)*1/totacc10

# MLP_W10= accuracy_score(test_labels10, predMLP10)*1/totacc10


for i in range(10000):

   if i < 10:

      X, y = stream.next_sample()

      HT.partial_fit(X, y, classes=[0, 1])

      NB.partial_fit(X, y, classes=[0, 1])

      MLP.partial_fit(X, y, classes=[0, 1])

   else:

      X, y = stream.next_sample()


      yvec10.append(y)


      prob_0 = (HT_W10 * HT.predict_proba(X)[0][0]) + (NB_W10 * NB.predict_proba(X)[0][0]) +
(MLP_W10 * MLP.predict_proba(X)[0][0])

      prob_1 = (HT_W10 * HT.predict_proba(X)[0][1]) + (NB_W10 * NB.predict_proba(X)[0][1]) +
(MLP_W10 * MLP.predict_proba(X)[0][1])


      if prob_1 > prob_0:
            pred_weighted10.append(1)
        else:
            pred_weighted10.append(0)

        HT.partial_fit(X, y)
        NB.partial_fit(X, y)
        MLP.partial_fit(X, y)

accurate10 = accuracy_score(yvec10,pred_weighted10)
print("Accuracy of Weighted Majority Voting Online Ensemble Classification using RBF Dataset 10: ",accurate10)

pred_weighted70 = []
yvec70 = []

HT_W70 = 0.2
NB_W70 = 0.3
MLP_W70 = 0.5

# totacc70 = accuracy_score(test_labels70, predHT70) + accuracy_score(y_test70, predNB70) + accuracy_score(y_test70, predMLP70)

# HT_W70 = accuracy_score(test_labels70, predHT70)*1/totacc70
# NB_W70 = accuracy_score(test_labels70, predNB70)*1/totacc70
# MLP_W70= accuracy_score(test_labels70, predMLP70)*1/totacc70

for i in range(10000):
    if i < 10:
        X, y = stream.next_sample()
        HT.partial_fit(X, y, classes=[0, 1])
        NB.partial_fit(X, y, classes=[0, 1])
        MLP.partial_fit(X, y, classes=[0, 1])
    else:
        X, y = stream.next_sample()

        yvec70.append(y)

        prob_0 = (HT_W70 * HT.predict_proba(X)[0][0]) + (NB_W70 * NB.predict_proba(X)[0][0]) +
(MLP_W70 * MLP.predict_proba(X)[0][0])
        prob_1 = (HT_W70 * HT.predict_proba(X)[0][1]) + (NB_W70 * NB.predict_proba(X)[0][1]) +
(MLP_W70 * MLP.predict_proba(X)[0][1])

        if prob_1 > prob_0:
            pred_weighted70.append(1)
        else:
            pred_weighted70.append(0)

        HT.partial_fit(X, y)
        NB.partial_fit(X, y)
        MLP.partial_fit(X, y)

accurate70 = accuracy_score(yvec70,pred_weighted70)
print("Accuracy of Weighted Majority Voting Online Ensemble Classification using RBF Dataset 70:
",accurate70)

#%%Hoeffding Tree Batch Classification
HT = HoeffdingTree()

data = pd.read_csv("RBF Dataset.csv")
data_np = data.to_numpy()
x = data_np[:,:10]
y = data_np[:,-1:]
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.5, random_state=42)


HT.fit(x_train, y_train.ravel())

predHT = HT.predict(x_test)

print("Accuracy score for Hoeffding Tree Batch Classification for RBF Dataset: ",accuracy_score(y_test, predHT))


data = pd.read_csv("RBF Dataset 10.csv")

data_np = data.to_numpy()

x10 = data_np[:,:10]

y10 = data_np[:,-1:]


x_train10,x_test10,y_train10,y_test10 = train_test_split(x10,y10,test_size=0.5, random_state=42)


HT.fit(x_train10, y_train10.ravel())

predHT10 = HT.predict(x_test10)

print("Accuracy score for Hoeffding Tree Batch Classification for RBF Dataset 10: ",accuracy_score(y_test10, predHT10))


data = pd.read_csv("RBF Dataset 70.csv")

data_np = data.to_numpy()

x70 = data_np[:,:10]

y70 = data_np[:,-1:]


x_train70,x_test70,y_train70,y_test70 = train_test_split(x70,y70,test_size=0.5, random_state=42)


HT.fit(x_train70, y_train70.ravel())

predHT70 = HT.predict(x_test70)

print("Accuracy score for Hoeffding Tree Batch Classification for RBF Dataset 70: ",accuracy_score(y_test70, predHT70))


#Naive Bayes Batch Classification
```

```
NB = GaussianNB()


predNB = NB.fit(train_features, train_labels).predict(test_features)

print("Accuracy score for Naive Bayes Batch Classification for RBF Dataset: ",accuracy_score(test_labels, predNB))


predNB10 = NB.fit(train_features10, train_labels10).predict(test_features10)

print("Accuracy score for Naive Bayes Batch Classification for RBF Dataset 10: ",accuracy_score(test_labels10, predNB10))


predNB70 = NB.fit(train_features70, train_labels70).predict(test_features70)

print("Accuracy score for Naive Bayes Batch Classification for RBF Dataset 70: ",accuracy_score(train_labels70, predNB70))


#Multilayer Perceptron with Batch Learning

MLP = MLPClassifier(hidden_layer_sizes = [200,200,200,200],

                    max_iter = 10000,

                    activation = 'tanh',

                    batch_size= 100,

                    solver = 'adam', random_state = 42)




MLP.fit(train_features, train_labels.ravel())

predMLP = MLP.predict(test_features)

print("Accuracy score for Multilayer Perceptron with Batch Classification for RBF Dataset: ",accuracy_score(test_labels, predMLP))


MLP.fit(train_features10, train_labels10.ravel())

predMLP10 = MLP.predict(test_features10)

print("Accuracy score for Multilayer Perceptron with Batch Classification for RBF Dataset 10: ",accuracy_score(test_labels10, predMLP10))
```

```
MLP.fit(train_features70, train_labels70.ravel())

predMLP70 = MLP.predict(test_features70)

print("Accuracy score for Multilayer Perceptron with Batch Classification for RBF Dataset 70:
",accuracy_score(test_labels70, predMLP70))


#Ensemble Learning with Majority Voting for Batch Classification

correctness = []


for i in range(5000):

  if ((predHT[i] + predNB[i] + predMLP[i]) < 2):

     predicted = 0

  else:

     predicted = 1


  if predicted == y_test[i]:

    correctness.append(1)

  else:

    correctness.append(0)


accuracy = [sum(correctness[:i])/len(correctness[:i]) for i in range(1,5000)]

print("Accuracy of Majority Voting Batch Ensemble Classification using RBF Dataset: ",accuracy[-1])


correctness10 = []


for i in range(5000):

   if ((predHT10[i] + predNB10[i] +predMLP10[i]) < 2):

      predicted10 = 0

   else:

      predicted10 = 1


   if predicted10 == y_test[i]:
      correctness10.append(1)
    else:
      correctness10.append(0)

accuracy10 = [sum(correctness10[:i])/len(correctness10[:i]) for i in range(1,5000)]
print("Accuracy of Majority Voting Batch Ensemble Classification using RBF Dataset 10: ",accuracy10[-1])

correctness70 = []

for i in range(5000):
    if ((predHT70[i] + predNB70[i] +predMLP70[i]) < 2):
        predicted70 = 0
    else:
        predicted70 = 1

    if predicted70 == y_test[i]:
      correctness70.append(1)
    else:
      correctness70.append(0)

accuracy70 = [sum(correctness70[:i])/len(correctness70[:i]) for i in range(1,5000)]
print("Accuracy of Majority Voting Batch Ensemble Classification using RBF Dataset 70: ",accuracy70[-1])

#Ensemble Learning with Weighted Majority Voting for Batch Classification
prediction_W1 = HT.predict_proba(x_test)
prediction_W2 = NB.predict_proba(x_test)
prediction_W3 = MLP.predict_proba(x_test)

totacc = accuracy_score(y_test, predHT) + accuracy_score(y_test, predNB) + accuracy_score(y_test, predMLP)
```

```
HT_W = accuracy_score(y_test, predHT)*1/totacc

NB_W = accuracy_score(y_test, predNB)*1/totacc

MLP_W = accuracy_score(y_test, predMLP)*1/totacc


pred_weighted = []

for i in range (len(x_test)):

   prob_0 = (HT_W * prediction_W1[i][0]) + (NB_W * prediction_W2[i][0]) + (MLP_W *
prediction_W3[i][0])

   prob_1 = (HT_W * prediction_W1[i][1]) + (NB_W * prediction_W2[i][1]) + (MLP_W *
prediction_W3[i][1])

   if prob_1 > prob_0:

      pred_weighted.append(1)

   else:

      pred_weighted.append(0)


accurate = accuracy_score(y_test,pred_weighted)

print("Accuracy of Weighted Majority Voting Batch Ensemble Classification using RBF Dataset:
",accurate)


prediction_W1 = HT.predict_proba(x_test10)

prediction_W2 = NB.predict_proba(x_test10)

prediction_W3 = MLP.predict_proba(x_test10)


totacc10 = accuracy_score(y_test10, predHT10) + accuracy_score(y_test10, predNB10) +
accuracy_score(y_test10, predMLP10)


HT_W10 = accuracy_score(y_test10, predHT10)*1/totacc10

NB_W10 = accuracy_score(y_test10, predNB10)*1/totacc10

MLP_W10 = accuracy_score(y_test10, predMLP10)*1/totacc10


pred_weighted10 = []
```

```
for i in range (len(x_test)):
    prob_0 = (HT_W10 * prediction_W1[i][0]) + (NB_W10 * prediction_W2[i][0]) + (MLP_W10 *
prediction_W3[i][0])
    prob_1 = (HT_W10 * prediction_W1[i][1]) + (NB_W10 * prediction_W2[i][1]) + (MLP_W10 *
prediction_W3[i][1])
    if prob_1 > prob_0:
        pred_weighted10.append(1)
    else:
        pred_weighted10.append(0)


accurate10 = accuracy_score(y_test10,pred_weighted10)
print("Accuracy of Weighted Majority Voting Batch Ensemble Classification using RBF Dataset 10:
",accurate10)


prediction_W1 = HT.predict_proba(x_test70)
prediction_W2 = NB.predict_proba(x_test70)
prediction_W3 = MLP.predict_proba(x_test70)


totacc70 = accuracy_score(y_test70, predHT70) + accuracy_score(y_test70, predNB70) +
accuracy_score(y_test70, predMLP70)


HT_W70 = accuracy_score(y_test70, predHT70)*1/totacc70
NB_W70 = accuracy_score(y_test70, predNB70)*1/totacc70
MLP_W70 = accuracy_score(y_test70, predMLP70)*1/totacc70


pred_weighted70 = []
for i in range (len(x_test)):
    prob_0 = (HT_W70 * prediction_W1[i][0]) + (NB_W70 * prediction_W2[i][0]) + (MLP_W70 *
prediction_W3[i][0])
    prob_1 = (HT_W70 * prediction_W1[i][1]) + (NB_W70 * prediction_W2[i][1]) + (MLP_W70 *
prediction_W3[i][1])
    if prob_1 > prob_0:
        pred_weighted70.append(1)
    else:
        pred_weighted70.append(0)


accurate70 = accuracy_score(y_test70,pred_weighted70)
print("Accuracy of Weighted Majority Voting Batch Ensemble Classification using RBF Dataset 70:
",accurate70)
```