Project 5 Report: Comparing Performance Of Contemporary Online Single And Ensemble Classifiers For Classification Of Data Streams

Fahad Waseem Butt, 21801356
Bilkent University, waseem.butt@ug.bilkent.edu.tr

The aim of this project is to effectively classify a stream of data that is continuously entering the system by employing scikit-multiflow framework that supports data mining using Python programming language.

1 INTRODUCTION

Over the last decade with the advancement in sensor technology and increase in data storage capability at a reduced cost, huge data sets are being saved that need to be efficiently analyzed. Data mining techniques cater for such challenges that involve limitations imposed by time and memory on various data streams. Data mining is a very useful methodology through which interesting structures buried within massive data can be established and represented in the form of graphs, trees, sets of equations, rules etcetera. Trends and patterns in data are identified by employing single or multiple algorithms to generate models to be applied to new situations [1]. Data mining methodology allows to train computers to make rapid decisions on large data sets which is not possible for a human [2]. However, if patterns are identified within data it is of a great benefit to differentiate, categorize and predict by humans [3].

Python provides a rich set of libraries that support data mining applications [2]. These libraries can be used to generate hyperplane datasets on which various algorithms can be applied and their effectiveness determined. The prime objective of providing Python interface and libraries is to offer ease of use as compared to shift focus on speed of execution [4]. This project involves learning scikit-multiflow and applying various data mining techniques. This is a useful learning platform that enables the implementation of complex methods with ease. For this assignment, this source was used for understanding the implementation [5].

2 DATASET GENERATION

In this part, the requirement was to generate hyperplane data with 20,000 instances using the Hyperplane generator. The dataset was to include 10 features with additional requirements of noise and the number of drifting features varying as per further requirements. For this purpose, the hyperplane generator function, “skmultiflow.data.HyperplaneGenerator” [6], was studied and the aforementioned requirement was met. Four data streams were generated, based on the instructions, and they were as follows.

- The first one with 10% noise and 2 drifting features. To meet the requirement, this was also saved as “Hyperplane dataset 10_2”. This may be referred to in short as Hyperplane dataset 10_2.
- Second one with 30% noise and 2 drifting features. This may be referred to in short as Hyperplane dataset 30_2.
- Third one with 10% noise and 5 drifting features. This may be referred to in short as Hyperplane dataset 10_5.
- Fourth one with 30% noise and 5 drifting features. This may be referred to in short as Hyperplane dataset 30_5.

These data streams were required as test inputs for the classifiers. These data streams with varying noise and drifting features form a good test base to check the performance of the classifiers.

3 DATA STREAM CLASSIFICATION WITH THREE SEPARATE ONLINE SINGLE CLASSIFIERS (HT, KNN, NB)

In this part, the requirement was to carry out datastream classification with three online single classifiers, i.e. Hoeffding Tree, K-Nearest Neighbors, Naive Bayes. A Hoeffding Tree (HT) Classifier assumes that data is not changing over time and, with this assumption, is able to learn from large data streams [7]. The K-Nearest Neighbors (KNN) Classifier has a simple working method in that it calculates the distance between new and old data points and makes a prediction on the closest K data points [8]. The Naive Bayes (NB) Classifier essentially makes use of Bayes Theorem to find probability and make predictions [9]. In this assignment, the following libraries were made use of to use these classifiers: "skmultiflow.trees.HoeffdingTreeClassifier" [10], "skmultiflow.lazy.KNNClassifier" [11], "skmultiflow.bayes.NaiveBayes" [12]. These models were then evaluated using the built-in function "skmultiflow.evaluation.EvaluatePrequential" [13]. The simulation was carried out by selecting one hyperplane at a time and assigning it as "sel_stream". The results, i.e. accuracy, kappa, training time, testing time, total time, size, were displayed for each classifier. Figure 1 shows the data gathered when comparing the classifiers on these parameters using Hyperplane dataset 10_2.

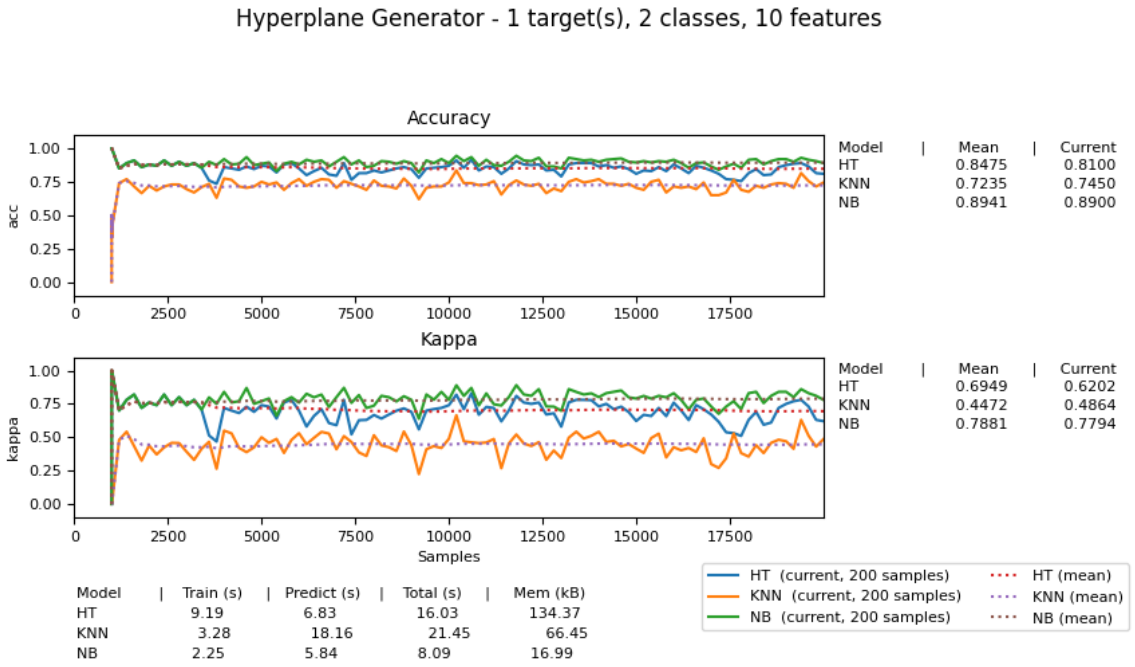

Figure 1: Performance of HT, KNN and NB Classifiers using Hyperplane dataset 10_2

It was observed that overall, the NB classifier outperformed the other classifiers in these parameters, and this can be understood better with the data from Figure 1 organized in Table 1 as follows.

Parameter	Hoeffding Tree	KNN	Naive Bayes
Accuracy (%)	84.75	72.35	89.41
Kappa	0.6949	0.4472	0.7881
Training Time (s)	9.19	3.28	2.25
Testing Time (s)	6.83	18.16	5.84
Total Time (s)	16.03	21.45	8.09
Size (kB)	134.3701	66.4492	16.9912

Table 1: Table of Performance of HT, KNN and NB Classifiers for Hyperplane dataset 10_2

In Table 1, it can be seen that the Naive Bayes Classifier has better accuracy and smaller run time than the other classifiers; i.e. the NB classifier is the most accurate and has the highest efficiency. Figures 2, 3 and 4 show the performance of the classifiers using Hyperplane dataset 30_2, Hyperplane dataset 10_5 and Hyperplane dataset 30_5 respectively.

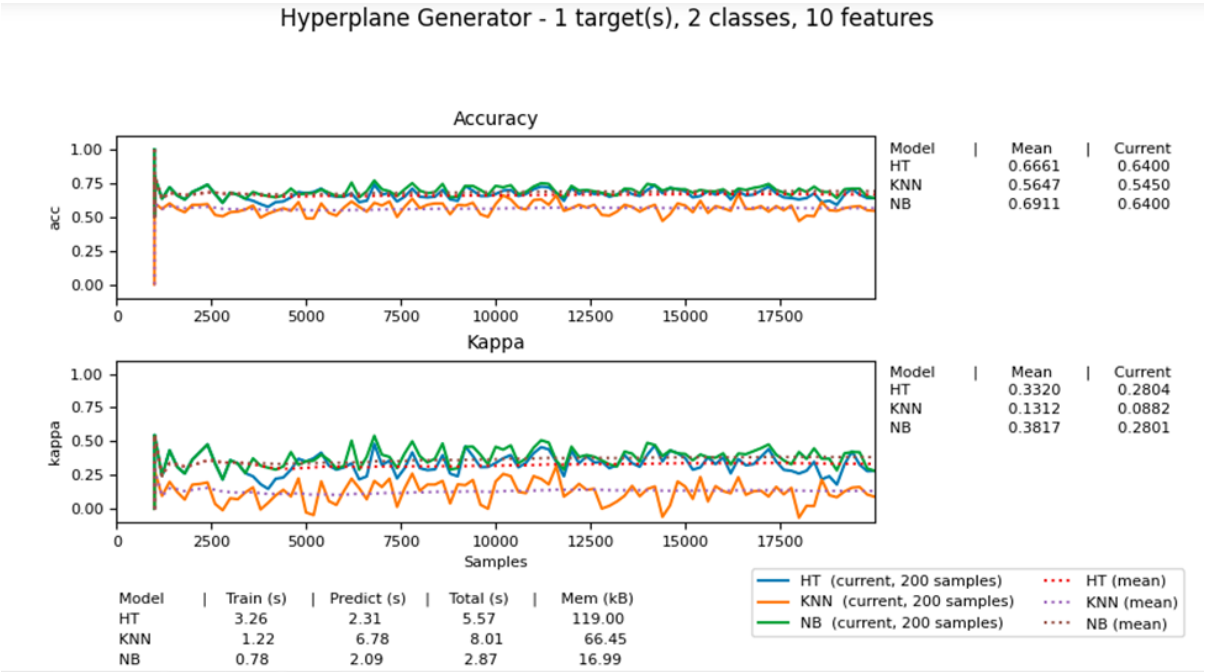

Figure 2: Performance of HT, KNN and NB Classifiers using Hyperplane dataset 30_2

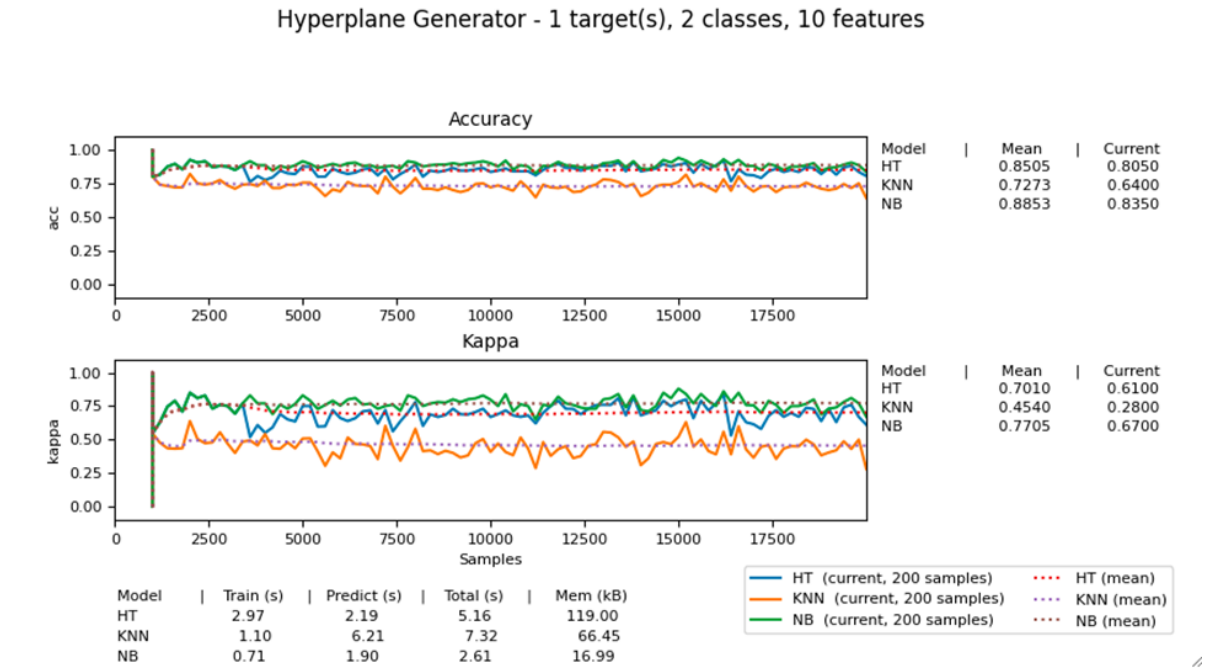

Figure 3: Performance of HT, KNN and NB Classifiers using Hyperplane dataset 10_5

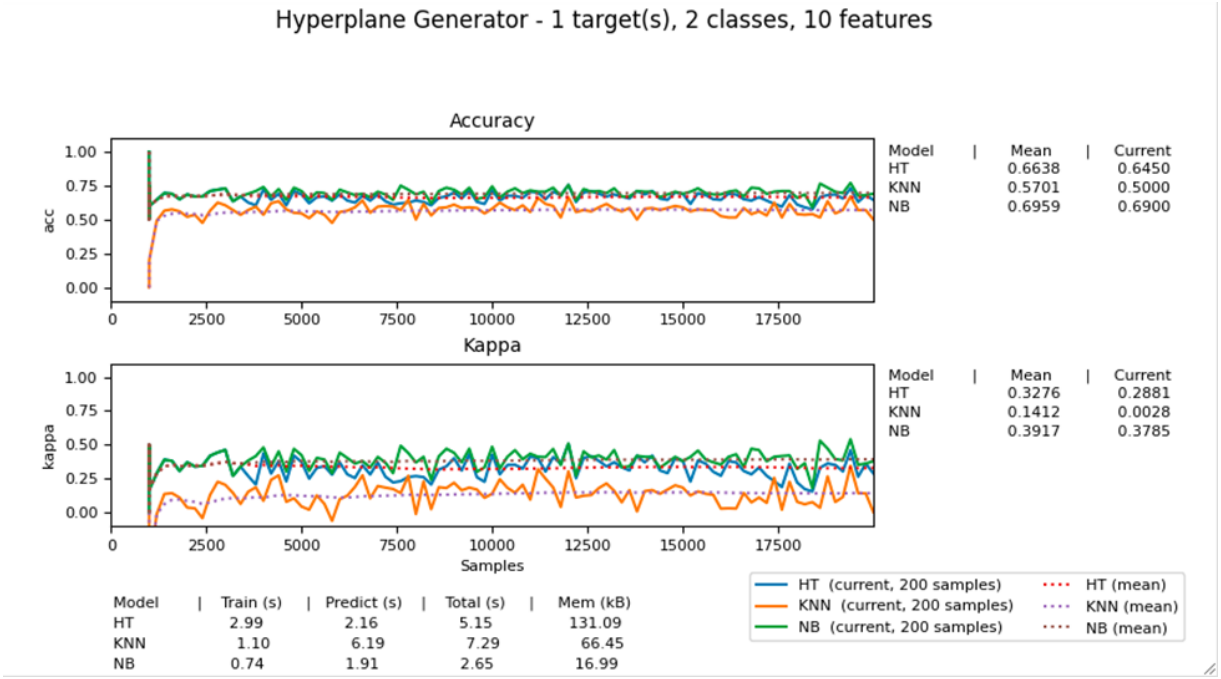

Figure 4: Performance of HT, KNN and NB Classifiers using Hyperplane dataset 30_5

When comparing Figures 1, 2, 3 and 4, it can be noted that a 10% noise percentage gave better accuracy. Increasing the noise percentage, for example to 30% caused the accuracy and kappa values to drop. It was observed that increasing the drifting feature caused the runtime to decrease, with the runtime of 5 drifting features being less than half that of the runtime with 2 drifting features. However this distinction is more noticeable for a 10% noise percentage, whereas it is difficult to understand when the noise percentage is higher.

The simulation was run with batch sizes 1, 100, and 1000. Table 2 below summarises the performance of the methods, in terms of accuracy and runtime for each data stream.

Parameter	Batch Size = 1	Batch Size = 100	Batch Size = 1000
HT Accuracy (%)	84.75	85.43	79.05
HT Runtime (s)	16.03	8.06	8.26
KNN Accuracy (%)	72.35	72.97	71.89
KNN Runtime (s)	21.45	1.94	1.65
NB Accuracy (%)	89.41	88.88	75.64
NB Runtime (s)	8.09	3.40	3.30

Table 2: Table of Performance Comparison with Batch Sizes 1, 100 and 1000

It can be observed from Table 2 that as the batch size is increased from 1 to 100, the runtime decreases, whereas there is no significant change in accuracy. As the batch size is increased from 100 to 1000, there is no significant decrease of runtime, however, there is a noticeable decrease in the accuracy. Hence increasing the batch size is helpful as it takes a chunk of data and processes it together for a faster runtime, but increasing it by too much makes the chunks too big and starts to cause a loss in accuracy.

4 DATA STREAM CLASSIFICATION WITH TWO ONLINE ENSEMBLE CLASSIFIERS (MV, WMV)

In this part there was a requirement to create an online ensemble classifier using the three classifiers (HT, KNN and NB). Two voting methods i.e. Majority Voting (MV) and Weighted Majority Voting (WMV) were required to be implemented from scratch in Python. Some assistance in writing the code for the voting methods was taken from this source [7].

The requirement was met by first finding the predictions of HT, KNN and NB. Later, the voting logic was implemented. In the case of the Majority Voting method, the prediction output of majority classifiers was used. For instance, if two or all classifiers predicted the same output, then this output was selected as the correct prediction (i.e. the majority). np.argmax(np.bitcount())[2] function was able to achieve the objective. Prediction of the classifier which is different from the other two classifiers, i.e. the minority, was rejected.

The weighted majority voting classifier logic was similarly implemented. The same function np.argmax(np.bitcount()) was utilised with an additional feature of assigning weights. From the prediction accuracy as observed earlier for each classifier it was found out that the NB classifier was more accurate. It was logical to assign a higher weight value to this classifier.

Similarly weights for other classifiers were assigned as per their accuracy. There is one thing to be noted while assigning weights; as the value of weight assignment increases, the voting method tends to converge towards the higher assigned weight classifier. For example, if a considerably larger value is assigned to the NB classifier, exact prediction accuracy of NB is achieved even though a form of majority voting classifier is being used. Therefore, an appropriate value of weight should be assigned.

Generally, it was observed that voting methods resulted in better prediction accuracy but did not exceed the accuracy of the classifier with best prediction. This is because there are cases when the best prediction accuracy classifier is the only one giving the correct prediction. In case of majority voting it will be discarded, being alone. However, in the case of the weighted voting method, there are chances that the best prediction accuracy classifier output will be the output of the algorithm. It is observed that weighted majority voting prediction accuracy is a little higher than simple majority voting as can be seen in Table 3.

	Majority Voting	Weighted Majority Voting
Accuracy (%), dataset 10_2	87.285	88.02
Accuracy (%), dataset 30_2	67.7	68.12
Accuracy (%), dataset 10_5	87.465	88.1
Accuracy (%), dataset 30_5	68.48	68.99

Table 3: Accuracy of Majority Voting and Weighted Majority Voting Classifiers

Ensemble methods are generally better than individual models but not necessarily. This is because a voting method is being used, in which any method which is not accurate will not outweigh the final prediction, since chances are that other two methods are giving correct predictions. As mentioned prior, ensemble methods are better than individual methods but not better than the best. It is not possible to get prediction accuracy better than the best in this case as only three classifiers are used, out which prediction accuracy of two are quite close and one of them is lower. If the number of classifiers is very large, there may be a possibility that output of the ensemble classifier is better than the individual models, but this has not been tested as it was out of scope of the current project.

It can also be observed that even with the accuracies for Majority Voting and Weighted Majority Voting, when the noise percentage is high, the accuracy drops.

When comparing all online and batch models (single and ensemble) in terms of their overall accuracies, as in Table 4, it is observed that the best performing model is still Naive Bayes, which has slightly better accuracy than Weighted Majority Voting. This may be because of some cases where an incorrect prediction by the other classifiers had more votes; this also explains the accuracy difference in Majority voting and Weighted Majority Voting better.

	MV	WMV	HT	KNN	NB
Accuracy (%)	87.285	88.02	84.75	72.35	89.41

Table 4: Accuracy of Online and Batch Voting Classifiers

When looking to improve the prediction accuracy of the online classifiers, a method to improve it in the case of K-Nearest Neighbors Classifier is to find the best value for K. Throughout this report before this point, K=10 was used. But when K=13 is used instead the accuracy improves from 72.85% to 74.30%. Another possible improvement may be to experiment with the weights for Weighted Majority Voting to find the best combination of weights, or even implement an adaptive weight learning algorithm. After improving the weights of the WMV Classifier to give even more weight to the Naive Bayes Classifier and reduced weight to the KNN Classifier, with the same weight for the Hoeffding Tree Classifier, the accuracy improved from 88.02% to 88.755%. This new accuracy for the WMV Classifier is closer to the accuracy for the NB Classifier as it is getting more contribution from it.

5 REFERENCES:

[1] R. Roiger, Data Mining A Tutorial-Based Primer (Second Edition). 2016. [Online]. Available: ISBN 9781498763974

[2] D. Phillips, F. Romano, P. Vo. T.H, M. Czygan, R. Layton and S. Ra, Python: Real-World Data Science. Packt Publishing, 2016. [Online]. Available: ISBN 1786468417

[3] M. Squire, Mastering Data Mining with Python ' Find patterns hidden in your data. Packt Publishing, 2016. [Online]. Available: ISBN 1785889958

[4] L. P. Cavalheiro, “Analysis on Python Performance for Data Stream Mining,” Federal University of Paraná, 2020. [Online]. Available: https://acervodigital.ufpr.br/bitstream/handle/1884/71435/R%20-%20E%20-%20LUCCA%20PORTES%20CAVALHEIRO.pdf

[5] J. Montiel, "skmultiflow-demo/skmultiflow-demo.ipynb", GitHub, 2019. [Online]. Available: https://github.com/jacobmontiel/skmultiflow-demo/blob/master/skmultiflow-demo.ipynb

[6] "skmultiflow.data.HyperplaneGenerator", scikit-multiflow, 2020. [Online]. Available: https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.data.HyperplaneGenerator.html

[7] https://analyticsindiamag.com/a-beginners-guide-to-hoeffding-tree-with-python-implementation/

[8] S. Robinson, "K-Nearest Neighbors Algorithm in Python and Scikit-Learn", Stackabuse, 2021. [Online]. Available: https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/

[9] R. Gandhi, "Naive Bayes Classifier", Medium, 2018. [Online]. Available: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

[10] "skmultiflow.trees.HoeffdingTreeClassifier", scikit-multiflow, 2020. [Online]. Available: https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.trees.HoeffdingTreeClassifier.html

[11] "skmultiflow.lazy.KNNClassifier", scikit-multiflow, 2020. [Online]. Available: https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.lazy.KNNClassifier.html

[12] "skmultiflow.bayes.NaiveBayes", scikit-multiflow, 2020. [Online]. Available: https://scikit-multiflow.readthedocs.io/en/latest/api/generated/skmultiflow.bayes.NaiveBayes.html

[13] "skmultiflow.evaluation.EvaluatePrequential", scikit-multiflow, 2020. [Online]. Available: https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.evaluation.EvaluatePrequential.html