COMPX523 - Data Stream Mining

Assignment 1 Report

Name: Jeffrey Jose
ID: 1313512
COMPX523-20A Assignment One
01/05/2020

Data Overview:

The evaluations used to test the methods within this analysis are primarily run across two datasets: data_n30000 and covtype_numeric.

Data: data_n30000

This dataset consists of 30,000 observations which are split into 3 variables: x_0, x_1 and y. There are overall two binary classes within the data so each observation can belong to either 'group' 0 or 'group' 1.

Using RStudio, I have identified 20,000 of the total 30,000 observations belong to 'group' 0 and the other 10,000 observations belong to 'group' 1.

Data: covtype_numeric

The second dataset contains 581,012 observations which are split into 11 variables as shown below:

```
 [1] "Elevation"                     "Aspect"                        "Slope"
 [4] "Horizontal_Distance_To_Hydrology"  "Vertical_Distance_To_Hydrology"    "Horizontal_Distance_To_Roadways"
 [7] "Hillshade_9am"                 "Hillshade_Noon"                "Hillshade_3pm"
[10] "Horizontal_Distance_To_Fire_Points" "class"
```

Unlike the previous dataset, there are overall seven classes within this dataset and the values of the remaining variables determine the class in which a particular observation would belong to.

Using RStudio, I had partitioned the number of instances belonging to each class and the percentage of overall population belonging to each class as shown below:

Class	No. Of Observations	Percentage
1	211840	36.5%
2	283301	48.8%
3	35754	6.2%
4	2747	0.5%
5	9493	1.6%
6	17367	3.0%
7	20510	3.5%

As shown by the result, a large percentage of the observations belong to class 2 followed by class 1 whereas the least number of observations belong to class 4.

The prior information provided about the dataset helps us to identify certain characteristics within the results of our methods and help us to relate the result back to the structure of the dataset more effectively.

Experiment 1: kNN variations and other algorithms

The first experiment looks at the computational results of our stream using different methods that are implemented within scikit-multiflow package and the MyKNNClassifier that was implemented by myself as a part of this project. The dataset used for the purpose of this assignment was data_n30000.

The methods used in this experiment are as follows:

1. kNN Classifier
2. Hoeffding Tree
3. Naïve Bayes
4. kNN + weighted by 1/distance (MyKNNClassifier)
5. kNN + standardization (MyKNNClassifier)

Pre-evaluation Method:

For the purpose of this assignment, I have kept all KNN classifier parameters as the default ones. Therefore, the value of neighbours is 5, maximum window size is 1000 and the leaf size is 30. For more comparative results among the Hoeffding Tree classifier, I have also tested the measures of the evaluator using Hoeffding Tree where the leaf prediction is defined as majority classifier and naïve bayes.

The results are as follows:

	Acc.	Kappa	Kappa T	Kappa M	Recall 0	Recall 1	Total Time
kNN(k5, w1000, maj)	0.6830	0.2705	0.3080	0.0489	0.75	0.53	15.83
Hoeffding Tree	0.6674	0.0535	0.2740	0.0022	0.68	0.51	3.14
Naïve Bayes	0.6666	0.0000	0.2723	-0.0001	0.67	Ill-def	1.67
kNN(k5, w1000, weighted)	0.6870	0.2787	0.3168	0.0610	0.75	0.53	17.22
kNN(k5, w1000, standardized)	0.6736	0.2485	0.2876	0.0208	0.74	0.51	100.65
Hoeffding Tree (leaf = mc)	0.6661	0.0013	0.2711	-0.0018	0.67	0.40	2.38
Hoeffding Tree (leaf = nb)	0.669	0.0614	0.2729	0.0006	0.68	0.50	2.87

Note: Naïve Bayes is used as the baseline method.

From the results above, it is very evident that the method which resulted in the highest accuracy for classification is that of the k nearest neighbours which used weighting as a measure of majority voting as opposed to majority voting. In machine learning, weighted voting is depended on the distance of each neighbouring instances to x_u. In my implementation, each $x_u \in neighbours(x_u, k)$ weights it's votes in favour of y_n (ie. The label associated with x_n) with 1 / $distance(x_n, x_u)$.

A keynote that I observed was that there happens to be a correlation/relationship between accuracy and kappa. As the weighted k nearest neighbours classifier has the highest accuracy, it also displays the highest kappa value. This is expected by the definition of both accuracy and kappa.

As the name implies, accuracy is the percentage of correctly classified instances out of all the instances observed.

$$Accuracy = \frac{Number\ Of\ Correct\ Predictions}{Total\ Number\ Of\ Predictions\ Made}$$

It is particularly useful on a binary classification such as the classification within the current dataset than classifying multi-class problems because it can be less clear exactly how the accuracy breaks down across those classes.

Kappa (Cohen's Kappa) on the other hand is like accuracy except that it is normalized at the baseline of random chance on our dataset. It is primarily used to control only those instances that may have been correctly classified by chance and is a much higher sensitive measure for justifying the predictive performance of streaming classifiers. Kappa is calculated as:

$$Kappa = \frac{(Total\ Accuracy - Random\ Accuracy)}{(1 - Random\ Accuracy)}$$

$$k = \frac{\Pr(a) - \Pr(\varepsilon)}{1 - \Pr(\varepsilon)}$$

The Kappa average is a useful measure on problems that have an imbalance in the classes). If the classifier is always correct, then the value of kappa will equate to 1 and if it's predictions coincide with the correct ones as often as those of a chance classifier then k equates to 0.

Kappa has two subset categories known as Kappa T and Kappa M:

Kappa M:

Kappa M is a measure that is used to compare against a majority class classifier instead of a chance classifier. The equation is as shown below:

$$k_m = \frac{\Pr(a) - \Pr(m)}{1 - \Pr(m)}$$

Situations where the distribution of predicted classes is incredibly different to the distribution of the true classes, a majority class classifier can perform much better than a given classifier while the classifier has a positive k value. In our results, we see that the baseline classifier, Naïve Bayes and the Hoeffding tree where the leaf predictions is set to majority class, the kappa m value returns a negative value. This means that for both the Naïve Bayes and Hoeffding Tree classifiers, the classifier's performance is rated very low and don't rely that heavily on the accuracy.

Although accuracy is a much more useful metric for binary classification, based on the explanation given above, I believe that the appropriate general metric for this current data is Kappa M. This mainly due to the large imbalance between the two binary classes within the dataset. Of the total 30,000 instances, 20,000 belong to group 0 (the majority class) and only 10,000 belong to group 1 (the minority class).

A Hoeffding Tree is an incremental decision tree learner for large data streams. Hoeffding tree is unque as it waits for new instances to arrive rather than using again the instances that it has already seen.

Naïve Bayes however is a classification algorithm known for its very low computational time (as seen in the table). Naïve Bayes is proposed from the formula proposed for the posterior distribution under Bayesian statistics which follows the form:

$$Posterior \approx \frac{Prior\ \times Likelihood}{Evidence}$$

This tells that the posterior (or probability of an event) is updated based on our prior beliefs and after accounting for some evidence. Naïve Bayes itself is an incremental algorithm and therefore it is well suited for data streaming. But it does also assume that the variables within our dataset are independent of each other (ie. Change in one attribute does not affect another attribute) which in real life situations may not always be the case.

Commonalities between Hoeffding Tree and Naïve Bayes:

The nodes on each branch of the Hoeffding tree holds the values that are used for splitting variables. For variables that are not continuous, these values are the same that is required for computing the Naïve Bayes predictions. A primary reason for the heavy use of Hoeffding tree by data scientists nowadays is that it has a high guarantee of successful performance. Studies have shown that using a Naïve Bayes learner at the leaf nodes of the Hoeffding Tree yields a higher classification performance when compared to using majority class classifiers at the leaf nodes. Our table showing the accuracy between Hoeffding tree of both Naïve Bayes and majority class classifier is a proof of that. Although the accuracy isn't much better, we can see that the Naïve Bayes leaves had a higher accuracy (0.669) compared to that of the majority class classifier leaf accuracy (0.666). However, the kappa value of the Naïve Bayes leaves was much greater (~ 6.0%) compared to majority classifier leaves which was roughly 0.13%.

From my results, I have noticed that the Hoeffding Tree, Naïve Bayes and the Majority classifier have performed very identically. The accuracy among all three methods are highly similar with only a small margin of difference. The recall value for class 0 is also identical however the recall for class 1 under Naïve Bayes returns ill-defined which suggests that the classifier never voted for class 1 given the stream of data.

Feature standardisation is a common method used in stream mining to ensure that the probability among each instance is equal. The equation of standardisation is shown as below:

$$x'_i = \frac{(x_i - \hat{x}_i)}{\sigma_i}$$

Where the values of each feature in the data have a zero mean and unity standard deviation.

From the table above, I have noticed that standardizing the k nearest neighbour does not improve the accuracy result when compared to the accuracy result of the default k nearest neighbour which abides by majority voting and no standardization of data. The accuracy of the standardized kNN is 67.36% as compared to the non-standardized kNN classifier which had an accuracy of 68.30%. This characteristic carries over to the Kappa values, where the kappa values of the non-standardisation was larger than that of the standardised data (refer to the table). K nearest neighbours uses Euclidean distance as it's means of comparing instances. In order for all of the features to have equal importance when calculating the distance, the features must have the same range of values. This can't be done through standardisation. While standardisation may be best suited for other classifiers, this is not the case for k nearest neighbours or any other distance-based classifiers. In the current dataset that is being used for this experiment, we can see that the classification is binary. This is a key aspect of logistic regression (with binomial family) and logistic regression does not require standardisation as they are unaffected by it.

I did notice that the computational time of the standardised k nearest neighbours classifier was much greater than that of the traditional non-standardised k nearest neighbours classifier (100.65 secs compared to 15.83 secs). I assume this is due to the extra calculation that is done for every feature in the dataset and therefore standardisation slows down the computational performance too.

Therefore, standardization did not improve my results.

If I was more interested in correct predictions for the minority class (class 1), even if they come at the expense of more incorrect predictions for the majority class (class 0), the metric I would focus on is the recall for class 1 and the algorithm I would choose is the k nearest neighbours classifiers. Recall is defined as how many of the true positives were found.

$$Recall = \frac{TP}{(TP + FP)}$$

Experiment 2: Hyperparameter tuning for kNN

The aim of this experiment to try to obtain better results than the k nearest neighbour models tested in experiment 1 by hyper-tuning the parameters such as the k value (number of neighbours) and the size of the sliding window. In experiment, we had set the k value to 5 and the window size to 1000 for all 3 k nearest neighbour combination: one without standardisation and weighting, one with weighting alone and one with standardisation alone.

Based on accuracy, the k nearest model that included the weighting alone had the highest accuracy with a value of 0.6870 and a recall value of 0.75 for class 0 and 0.53 for class 1. The default k nearest model which included neither weighting or standardisation had exactly the same recall for both classes as the one that included weighting, but the accuracy was slightly lower with a value of 0.6830. The kappa values for the weighted model was much better than that of the default model with a value of 0.2787 as compared to 0.2705. Therefore, I will try different combinations of hyper-parameters to try improve on the current result. Although improvement is not guaranteed, the tests will help us identify the behaviour of the k nearest model with a changing combination of hyperparameters.

For this experiment I had decided to experiment using three levels for k = [1, 3, 20] and also three levels for the window size = [15, 600, 6000]. The first table shows the performance result of each combination when only using majority class voting:

	Acc.	**Kappa**	**Kappa T**	**Kappa M**	**Recall 0**	**Recall 1**	**Total Time**
KNN(k=1, w=15, maj)	0.8058	0.5870	0.5760	0.4173	0.90	0.67	6.61
KNN(k=1, w=600, maj)	0.7154	0.3458	0.3788	0.1462	0.77	0.57	11.73
KNN(k=1, w=6000, maj)	0.6318	0.1635	0.1964	-0.1045	0.72	0.44	77.96
KNN(k=3, w=15, maj)	0.7398	0.4652	0.4320	0.2193	0.88	0.57	6.96
KNN(k=3, w=600, maj)	0.7057	0.3285	0.3577	0.1172	0.77	0.56	12.28
KNN(k=3, w=6000, maj)	0.6399	0.1668	0.2139	-0.0804	0.71	0.45	77.89
KNN(k=20, w=15, maj)	0.6666	0.000	0.2723	-0.0001	0.67	Ill-def	1.45
KNN(k=20, w=600, maj)	0.6879	0.2485	0.3188	0.0637	0.74	0.54	13.34
KNN(k=20, w=6000, maj)	0.6601	0.1474	0.2580	-0.0198	0.70	0.48	80.84

The second table shows the predictive performance result for each combination when the classification technique is weighted class voting:

	Acc.	Kappa	Kappa T	Kappa M	Recall 0	Recall 1	Total Time
KNN(k=1, w=15, weight)	0.8058	0.5870	0.5760	0.4173	0.90	0.67	6.75
KNN(k=1, w=600, weight)	0.7154	0.3458	0.3788	0.1462	0.77	0.57	11.65
KNN(k=1, w=6000, weight)	0.6318	0.1635	0.1964	-0.1045	0.72	0.44	74.37
KNN(k=3, w=15, weight)	0.7733	0.5320	0.5051	0.3198	0.90	0.62	6.98
KNN(k=3, w=600, weight)	0.7120	0.3415	0.3713	0.1359	0.78	0.57	12.01
KNN(k=3, w=6000, weight)	0.6379	0.1776	0.2096	-0.0864	0.72	0.45	74.66
KNN(k=20, w=15, weight)	0.6666	0.000	0.2723	-0.0001	0.67	Ill-def	1.40
KNN(k=20, w=600, weight)	0.6990	0.3083	0.3430	0.0969	0.76	0.55	15.66
KNN(k=20, w=6000, weight)	0.6536	0.1721	0.2439	-0.0392	0.71	0.47	77.81

The final table shows the predictive performance for each combination when the data is standardized:

	Acc.	Kappa	Kappa T	Kappa M	Recall 0	Recall 1	Total Time
KNN(k=1, w=15, std)	0.7129	0.3776	0.3734	0.1387	0.81	0.56	21.78
KNN(k=1, w=600, std)	0.6978	0.3093	0.3403	0.0933	0.77	0.55	71.77
KNN(k=1, w=6000, std)	0.6336	0.1682	0.2001	-0.0994	0.72	0.45	2703.67
KNN(k=3, w=15, std)	0.6810	0.3318	0.3037	0.0430	0.81	0.52	21.26
KNN(k=3, w=600, std)	0.6921	0.2981	0.3280	0.0764	0.76	0.54	69.10
KNN(k=3, w=6000, std)	0.6425	0.1713	0.2196	-0.0726	0.72	0.46	518.05
KNN(k=20, w=15, std)	0.6666	0.000	0.2723	-0.0001	0.67	Ill-def	15.48
KNN(k=20, w=600, std)	0.6840	0.2358	0.3102	0.0519	0.73	0.54	69.70
KNN(k=20, w=6000, std)	0.6577	0.1390	0.2528	-0.0270	0.70	0.47	510.87

From the results that I have obtained above, I believe that the best model for this dataset is that the unstandardized and unweighted KNN where the value of k was 1 and the window size was 15. This classifier uses majority class voting. The predictive performance of this model is the exact same as the predictive performance of the weighted KNN model where the value of k is 1 and the value of window size is 15. The recall among these two models were the same too. However, I opted to choose the unweighted model as the computational time was slightly quicker with a value of 6.61 seconds as

compared to the weighted k nearest classifier where the computational time was 6.75 seconds. Since there is a relationship between kappa and accuracy, we do expect the kappa metric to reduce as the value of k increases and this is exactly what is observed through our results.

A key information that I have found through my experiments is that weighting does not affect a model if the value of k is less than 2. This is because there is only one distance to compare and therefore weighting it won't make a difference. Thus, the predictive performance of my models where k = 1 was exactly the same for both the weighted and unweighted model. Only the computational time differed slightly.

The impact of window length on the execution time of a model is very influential. From my results, it is obvious that the execution time of a model increases as the window size increases. The table below shows how the execution time increases as the window size increases for models that use majority class only, models that are weighted and the models that are standardized for where the value of k is 3.

	Majority Classifier	Weighted Classifier	Standardized
Window Size = 15	6.96	6.98	21.26
Window Size = 600	12.28	12.01	69.10
Window Size = 6000	77.89	74.66	518.05

Throughout this experiment, I tried three k values that range from low to high value so that I could observe the impact of a small k and a larger k. I observed that when k is small, I observed that the accuracy is higher given that the window size is kept constant. This pattern is repeated among models that use classifiers that are unweighted, weighted and standardized.

I also decided to choose a large range for the window size as I believe that the size of the window has a major impact on the predictive performance of a model, and I wanted to test this theory. From the results that I have produced, I do see a very faint pattern that given the k value is kept constant, the performance of the model decreases on most cases as the window size increases.

Experiment 3: Real data

For this final experiment, I will be using the "covtype_numeric" dataset which holds more than 500,000 instances. The aim of this experiment is to identify the performance of each classifier on a real data. The methods used within this experiment are:

1. KNN Classifier
2. Hoeffding Tree
3. Naïve Bayes
4. KNN + weighted by 1/distance
5. KNN + standardisation
6. Hoeffding Adaptive Tree Classifier
7. OzaBaggingAdwin
8. Adaptive Random Forest

The results are shown in the table below:

	Accuracy	Kappa	Kappa T	Kappa M	Time
Knn(k = 5, w = 1000, maj)	0.8509	0.7947	0.3934	0.7171	45.45
Knn(k = 10, w = 1000, maj)	0.8251	0.7602	0.2883	0.6681	45.00
Knn(k = 10, w = 1000, weighted)	0.8556	0.8021	0.4127	0.7261	47.52
Knn(k = 5, w = 1000, std)	0.6325	0.4927	-0.4952	0.3027	434.88
Knn(k = 10, w = 1000, std)	0.6280	0.4878	-0.5133	0.2942	435.20
Hoeffding Tree	0.7334	0.6376	-0.0845	0.4942	16.36
Hoeffding Tree (lp = majority class)	0.6233	0.4537	-0.5327	0.2852	6.45
Hoeffding Tree (lp = naïve bayes)	0.7087	0.6100	-0.1853	0.4472	12.68
Naïve Bayes	0.6987	0.5745	-0.2258	0.4283	13.85
Hoeffding Adaptive Tree Classifier	0.7608	0.6708	0.0175	0.5451	35.63
OzaBaggingAdwin	0.8553	0.8017	0.4135	0.7265	1194.11
Adaptive Random Forest	0.8549	0.7991	0.4096	0.7246	281.84

The dataset has 7 classes and therefore the recall for each class under the various methods had to be separately recorded:

	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7
Knn(k = 5, w = 1000, maj)	Ill-Def	0.79	0.92	0.68	0.81	0.80	0.76	0.88
Knn(k = 10, w = 1000, maj)	Ill-Def	0.75	0.92	0.62	0.77	0.75	0.71	0.85
Knn(k = 10, w = 1000, weighted)	Ill-Def	0.81	0.93	0.71	0.79	0.78	0.75	0.86
Knn(k = 5, w = 1000, std)	Ill-Def	0.49	0.83	0.31	0.59	0.43	0.36	0.51
Knn(k = 10, w = 1000, std)	Ill-Def	0.49	0.84	0.30	0.56	0.41	0.34	0.50
Hoeffding Tree	0.0	0.54	0.85	0.59	0.77	0.63	0.64	0.82
Hoeffding Tree (lp = majority class)	Ill-Def	0.47	0.70	0.35	0.61	0.37	0.46	0.76
Hoeffding Tree (lp = naïve bayes)	0.00	0.53	0.85	0.59	0.77	0.50	0.64	0.85
Naïve Bayes	Ill-Def	0.55	0.79	0.54	0.69	0.54	0.51	0.78
Hoeffding Adaptive Tree	0.0	0.62	0.86	0.59	0.80	0.67	0.65	0.82
OzaBaggingAdwin	Ill-Def	0.81	0.93	0.70	0.81	0.79	0.74	0.87
Adaptive Random Forest	Ill-Def	0.80	0.91	0.73	0.84	0.81	0.78	0.85

The no-change classifier is not available in scikit-multiflow and therefore Naïve Bayes will be the baseline.

Out of the models that were tested during this experiment, the weighted k nearest neighbour classifier with a k value of 10 and a maximum window size of 1000 achieved the best result in terms of accuracy

with a value of 85.56%. Although accuracy is the prime metric to compare multiple sets of models, evaluation does not always consider accuracy alone as seen. The major focus of evaluation of models is to recognise how flexible a particular model is which in turn means how will it perform on a different dataset. The second metric that should be taken into account is the recall. As mentioned previously in the report, recall is a measure of how great our model is when all the actual values are positive. A recall value of zero means that the model is broke and cannot get one prediction right when the actual value was positive (FP) and a recall value of one means that the model is well enough to correctly predict when the actual value was yes[1]. Especially in this case, accuracy should not be the only dependent metric due to the large imbalance within our dataset. Generally, with recall we do not want a low value, but we also do not want 1. If the recall is one, it could mean that our model has correctly classified all the values as positive when it was indeed the actual value. But it could also mean that our model has just classified every value as positive which in turn means that it has low precision.

The no-change classifier is one of the simplest classifiers for data streams in which it predicts the label for a new instance to be the true label of the instance. It does not require instance features and therefore it is very easy to implement. The classifier only makes mistakes on the boundary cases if long sequences of "no intrusion" is proceeded with a briefer period of "intrusion" but it also adjusts quickly to the label patterns.

In our experimental case, Naïve Bayes was treated as the baseline case since no-change classifiers are note available in scikit-multiflow.

Accuracy:

The baseline classifier, Naïve Bayes, had an accuracy of 69.87%. Since this is the baseline classifier, this should be the lowest accuracy we would expect from most models. However, in my experiment, I have found 3 other models that have performed worse than Naïve Bayes in terms of accuracy. Both k nearest neighbours with standardisation have yielded an accuracy of 63.25% and 62.80% and the Hoeffding Tree with the majority class classifier as the leaf predictor yielded an accuracy score of 62.33%.

It is a concern when a model is outperformed by the no-change classifier as the no-change classifier is the baseline model. The skill of this model provides the foundation for the lowest acceptable performance of a machine learning model on our specific dataset. If a model achieves a performance that is lower than that of the baseline classifier, something is wrong with the implementation. For example, in this case, we see that both the standardised k nearest neighbour models results in performance worse than Naïve Bayes which makes me wonder if standardising the data was the most appropriate thing to do for this particular dataset.

A large majority of my models have an accuracy that is above the baseline model which had an accuracy of 69.87%. This then carried over to the kappa where apart from the 3 models that had an accuracy worse than the baseline, all other models had a kappa value that was above that of the baseline Naïve Bayes model (0.5745). However, the computation time varied across different models based on the classifier that was being used. KNN models took the longest time to execute (even longer if it was standardized) and the Hoeffding Tree which used the majority class as the leaf predictor took the least time to execute with an execution time of 6.45 seconds. When assessing the recall among every class for each method, it was identified that generally most of the methods had higher recall for every class than the Naïve Bayes method. This is very promising as it tells us that our models do perform better than the ground truth model.

KNN Variants:

[1] https://towardsdatascience.com/precision-vs-recall-386cf9f89488

A large majority of the KNN variants performed incredibly well when compared with other models. I did notice that the accuracy for the weighted KNN model where k was equal to 10 and the window size was equal to 1000 gave me the highest accuracy and kappa value. This therefore indicated to me that this was by far the best model in the experiment. Both the KNN models which were standardized had accuracy values that were less than that of the baseline Naïve Bayes with KNN classifier where k = 5 yielding an accuracy score of 63.25% and the KNN classifier where k = 10 yielding a result of 62.80%. Both these models were unweighted, and this suggested to me that standardizing the data in this situation was not the best idea. I have noticed through this experiment, that the computational time of standardized model is far greater than those that are not standardized. This can be shown as both KNN models that were standardized had an execution time of 434.88 seconds and 435.20 seconds respectively. Even the recall among all classes are low for the models that were standardized. I do believe that the unstandardized KNN models do produce reasonable solution to this problem. In terms of accuracy when compared to other models, KNN has performed very well however the execution times of KNN models are much longer than those of Hoeffding Tree or Naïve Bayes (ignoring standardized KNN models used in the experiment). When looking at the recall, a large majority of classes have recall values above 70% when used with KNN variants but among other methods, they do seem to vary largely between the 50% to 70% region. This is why I fully believe that KNN provides a reasonable solution to this problem based on the predictive performance and processing time.

Ensemble Comments:

During this experiment, the two ensembles I tried was Oza Bagging Adwin and Adaptive Random Forest. The most important aspect oza bagging adwin that I noticed immediately, was it's slow execution time. In my experiment, I recorded a time of 1194.11 seconds for the method to compute. The oza bagging adwin is very good at capturing quick changes and gradual changes when the change is not very slow, but it has a big problem when the change is gradual and slow. The ensemble had a very good accuracy rate with a value of 85.53% and a kappa value of 0.8017. The recall among all classes seems to be a good value as most classes have between 0.7 and 0.92 as the recall value. This is a very good model for the current dataset, but the computation time is the only concern.

Adaptive Random Forest had a much faster execution time of 281.84 seconds and the predictive performance is almost just as good. According to the moa website, '*a practical and simple way to against an Adaptive Random Forest is to compare classification performance in terms of Accuracy/Kappa M and Kappa T'.* The accuracy rate was defined as 85.49% and the kappa value was 0.7991. This is a very excellent predictive performance and is backed up by the high recall values which are between 0.73 and 0.91 suggesting that the Adaptive Random Forest is a highly suitable classification model to use for this particular data-set and similar dataset.

Reference

Bifet, A. (n.d.). Machine Learning For Data Streams. Retrieved May 6, 2020, from https://www.cms.waikato.ac.nz/~abifet/book/

Bifet, A. (n.d.). Improving Adaptive Bagging Methods for Evolving Data Streams. Retrieved May 6, 2020, from https://www.cs.waikato.ac.nz/~ml/publications/2009/bifet_ACML09.pdf

Adaptive Random Forest. Retrieved May 7, 2020, from https://moa.cms.waikato.ac.nz/adaptive-random-forest/