# A Framework for Classification in Data Streams using Multi-strategy Learning

Ali Pesaranghader[1] (✉), Herna L. Viktor[1] and Eric Paquet[2]

[1] School of Electrical Engineering and Computer Science
University of Ottawa, Canada
{apesaran,hviktor}@uottawa.ca
[2] Information and Communications Technologies
National Research Council of Canada
eric-paquet@nrc-cnrc.gc.ca

**Abstract.** Adaptive online learning algorithms have been successfully applied to fast-evolving data streams. Such streams are susceptible to concept drift, which implies that the most suitable type of classifier often changes over time. In this setting, a system that is able to seamlessly select the type of learner that presents the current "best" model holds much value. For example, in a scenario such as user profiling for security applications, model adaptation is of the utmost importance. We have implemented a *multi-strategy* framework, the so-called TORNADO environment, which is able to run multiple and diverse classifiers simultaneously for decision making. In our framework, the current learner with the highest performance, at a specific point in time, is selected and the corresponding model is then provided to the user. In our implementation, we employ an Error-Memory-Runtime (EMR) measure which combines the *error-rate*, the *memory usage* and the *runtime* of classifiers as a performance indicator. We conducted experiments on synthetic and real-world datasets with the Hoeffding Tree, Naive Bayes, Perceptron, K-Nearest Neighbours and Decision Stumps algorithms. Our results indicate that our environment is able to adapt to changes and to continuously select the best current type of classifier, as the data evolve.

**Keywords:** Data Stream Mining, Classification, Concept Drift, Parallel Learning, Multi-strategy Learning, Adaptive Learning.

## 1 Introduction

Online and adaptive learning from evolving data streams have attracted the attention of many researchers during the last decade. Online learning algorithms continuously update their classification models by processing instances one-by-one to avoid any performance degradation [1, 15]. Adaptive learning algorithms not only update classification models online but also monitor any change in the data distribution (the so-called *concept drift*), by using drift detectors. Subsequently, the models are updated once a drift is detected [2-6]. Online and adaptive learning algorithms have been successfully applied in machine learning

applications, robotics, recommender systems, and intrusion detection systems [7-9].

Recently, there has been a surge of interest in so-called *pocket data mining*, where the aim is to run data mining models on small devices [13]. In this setting, the models produced by data stream mining algorithms should not only have low error-rates. Rather, the learning techniques should also be lightweight and efficient in terms of runtime and other resource allocations. Specifically, it has been shown that running even a lightweight decision tree classifier will often crash a mobile device [13, 14]. It then becomes important to consider issues such as storage space and memory utilization, when building models within a streaming environment.

However, in most streaming studies [2-9], the error-rate (or accuracy) is used as the defining measure of performance for evaluating adaptive learners. The interplay with memory usage and runtime considerations has only been investigated in a few studies. For example, Bifet et al. [10] considered memory, time and accuracy to compare the performances of ensembles of classifiers. Here, each measure was individually used for evaluation. In [11], Bifet et al. introduced the RAM-Hour measure, where every RAM-Hour equals to 1 GB of RAM occupied for one hour, to compare the performance of three versions of perceptron-based Hoeffding Trees. In addition, Zliobaite et al. [12] proposed the return on investment (ROI) measure for probing whether adaptation of a learning algorithm is beneficial. They concluded that adaptation should only take place if the expected gain in performance, measured by accuracy, exceeds the cost of other resources (e.g. memory and time) required for adaptation. In a recent study by Olorunnimbe et al. [13], the ROI measure is employed to adapt the number of classifiers used in an ensemble setting.

In this paper, we introduce the Tornado framework that addresses the above-mentioned issues. In our environment, we implemented a number of very diverse learners that are run in *parallel* against a single data stream. A dynamic, *multi-strategy learning* approach is followed, where diverse classifiers co-exist. All of the learners access the stream at the same time and proceed to construct their individual models in *parallel*. This framework thus stands in contrast to existing environments, such as MOA [15] and Weka [16], where the different types of classifiers are run independently and/or where an ensemble is employed. We also introduce an Error-Memory-Runtime (EMR) measure that is used to continuously assess and rank the performance of the different types of classifiers. This ranking is subsequently used to dynamically select the model constructed by the current "best" learner and present it to the users.

The remainder of this paper is organized as follows: The next section introduces the EMR measure that we use to select the "best" model at a specific time. We detail our Tornado framework in Section 3. Section 4 describes our experimental evaluation, and Section 5 concludes the paper.

## 2 Balancing Performance Measures

As stated above, error-rate is most often used to evaluate the performance of classifiers against a data stream that is susceptible to concept drift. Intuitively, as illustrated in Fig. 1, a change in the distribution of the data, as caused by concept drift, may lead to the error-rates of different types of classifiers to increase or decrease. In this figure, we simulate a number of drift points and show that, as a drift occurs, the classifier with the lowest error-rate changes. Based on this observation, it follows that a multi-strategy learning system where different types of classifiers co-exist and where the model, from the current "best" learner is provided to the users, may hold much value.

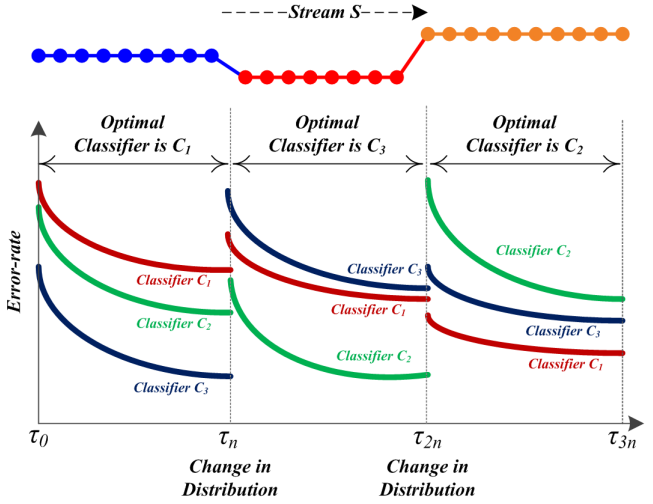



Fig. 1: Illustration of Concept Drift and Error-rate Interplay

However, following an "*error-rate-only*" approach is not beneficial in all settings. For instance, in an emergency response setting, the response time, i.e. the time it takes to present a model to the users, may be the most important criterion. That is, users may be willing to sacrifice accuracy for speed and partial information. Further, reconsider the area of pocket (or mobile) data mining, which has much application in areas such as defense and environmental impact assessment [14]. Here, the memory resources may be limited, due to connection issues, and thus reducing the memory footprint is also of importance.

In such a pocket data mining setting, consider two classifiers $C_1$ and $C_2$ which are used for classification over stream $S$. Suppose that the current model constructed by classifier $C_1$ has an 8.0% error-rate, 20 MB memory usage, and 100 Seconds runtime. On the other hand, the model constructed by classifier $C_2$ has the same error-rate, 1.5 MB memory usage, and 10'000 seconds runtime. It

follows that these two classifiers have the same level of error-rate. However, the memory usage of the first classifier may terminate the program, when running on a mobile device [14]. Further, the second classifier may not be suitable in an emergency response setting, where the goal is to optimize *just-in-time* decision making.

Based on these observations, we introduce the Error-Memory-Runtime (EMR) measure. The EMR measure is defined as:

$$EMR_C = \frac{w_e \cdot E_C + w_m \cdot M_C + w_r \cdot R_C}{w_e + w_m + w_r} \tag{1}$$

where $w_e$ is the error-rate cost weight, $E_C$ is the error-rate of classifier $C$, $w_m$ refers to the memory usage cost weight, $M_C$ denotes the memory usage of classifier $C$, $w_r$ depicts the runtime cost weight and $R_C$ refers to the runtime of classifier $C$. We define the *domain dependent* cost weights for error-rate, memory usage and runtime to emphasize their importance as we measure the cost of classifiers. In order to use the Equation (1), we normalize the values of error-rate, memory usage and runtime. A high EMR value means that the classifier has resulted in a high error-rate, high memory usage, and/or high runtime. Hence, a classifier with the lowest EMR is preferred in a multi-strategy learning setting.

In the real world, it follows that the values of the three weights ($w_e$, $w_m$, $w_r$) will be set to reflect the current domain of application. For instance, a classifier which has been shown to be very slow over the last period of time may be made to fade away. Alternatively, if the memory resources are limited, such as the case in a pocket data mining scenario [13], the value of $w_m$ may be set higher. On the other hand, if memory is abundant, but accuracy and speed of model construction are of importance, the $w_m$ value may be much lower (or even set to zero).

We use the EMR measure to score each classifier as follows. The score of classifier $C$ is calculated by Equation (2). Since the lowest EMR value is preferred, it follows that the highest score is preferred. Deduction of the lowest EMR from 1 results in the highest score.

$$Score_C = 1 - EMR_C \tag{2}$$

The average EMR which represents the average cost of a classifier over time is defined as follow:

$$\overline{EMR_C} = \frac{1}{T} \cdot \sum_{\tau=1}^{T} EMR_C{}^{\tau} \tag{3}$$

In this equation, $T$ is the total number of EMRs calculated for the classifier $C$ so far and $EMR_C{}^{\tau}$ is the $\tau^{\text{th}}$ EMR of that classifier. In addition, the average score of the classifier $C$ is defined as:

$$\overline{Score_C} = 1 - \overline{EMR_C} \tag{4}$$

## 3 The Tornado Framework

In this section, we introduce the TORNADO framework, coded in Python, as outlined in Fig. 2. Recall that, in our framework, a number of diverse classifiers are executed in *parallel*, against the same stream. In our system, a number of diverse base classifiers co-exist. That is, we include classifiers with different learning styles such as rule-based learners, decision trees, and nearest neighbour approaches in our Framework. This choice is based on the observation that employing diverse learning styles often leads to highly accurate and stable models.

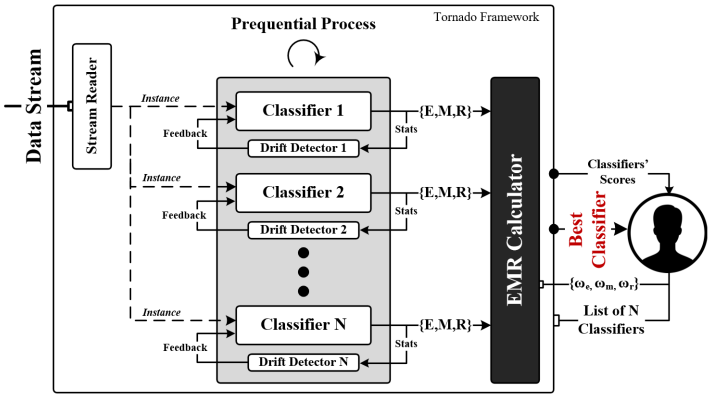



Fig. 2: The TORNADO Framework

We focus on the STREAM READER, CLASSIFIERS, DRIFT DETECTORS and EMR CALCULATOR components of the framework in this section. The inputs are a Stream, the error-rate weight $w_e$, the memory usage weight $w_m$, and the runtime weight $w_r$. Our framework operates based on the *prequential* approach where instances are first tested and then used for training, allowing for a maximum usage of the data [1 and 18].

The framework's workflow is as follows. Initially, the STREAM READER reads instances from the stream one-by-one, and sends each to the learners for model construction. Each learner builds incremental models. That is, it first tests the instance and then trains from the latter for updating its model. At the same time, CLASSIFIERS send their statistics (e.g. error-rate and total number of instances observed since the last drift), to their respective DRIFT DETECTOR in order to detect a potential occurrence of a drift. Also, each one of the classifiers sends the error-rate, memory usage and runtime (including testing and training time) to the EMR CALCULATOR in order to evaluate the scores which are subsequently ranked. As a result, the current model with the highest rank is presented to the user. This selection may change, as we learn incrementally from the stream and concept drift occurs. This process continues until either a pre-defined condition is met or when all the instances in the stream are observed.

The following observation is noteworthy. Our multi-strategy framework contains different types of base learners, i.e. it is different from an ensemble of classifiers. In this framework, the individual learners proceed independently to construct their models. The reason for this design decision is that we aim to utilize diverse learning strategies that potentially address concept drifts differently. However, future work may include incorporating lightweight ensembles, i.e. ensembles with lower memory utilization, into our Framework.

Fig. 3 gives an example showing how TORNADO runs the classifiers simultaneously and then recommends the best one by considering their scores at each time interval. During $\tau_0$ to $\tau_n$, classifier $C_1$ is considered as being the best classifier. The data distribution change at $\tau_n$. It follows that, in the interval in between $\tau_n$ to $\tau_{2n}$, classifier $C_3$ is the best classifier and is recommended to the user. Then, a drift occurs at $\tau_{2n}$ which results in classifier $C_2$ becoming the one with the highest score.

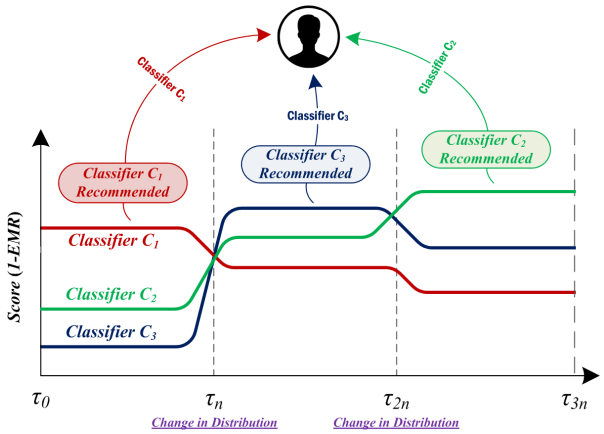



Fig. 3: Recommending the Classifier with the Highest Score

The pseudocode of our approach is shown in Algorithm 1. The inputs are a Stream, a list of Classifiers, the error-rate weight $w_e$, the memory usage weight $w_m$, and the runtime weight $w_r$. The outputs are the *scores of classifiers* and the *classifier recommended*. The first instance is used to train all classifiers. Future instances are tested and used for the training by the classifiers. If there is a drift, the classifier and the corresponding drift detector are reset. When an instance is processed, the CALCULATESCORES function is invoked in order to calculate the scores associated with the classifiers. Within this function, the NORMALIZELIST function normalizes the error-rate, the memory usage and the runtime. Subsequently, the CALCULATEEMR function calculates the EMRs associated with the various classifiers. Finally, all of the classifiers scores are shown to the users and the classifier with the current lowest EMR is recommended. Recall that this process continues until the process is terminated by the user, or the end of the stream is met.

**Algorithm 1** High-level Pseudocode of the TORNADO Framework

```
inputs:
    stream                                          ▷ Stream of Instances
    classifiers                                     ▷ List of Classifiers
    w_e, w_m, w_r                  ▷ Error-rate, Memory usage, Runtime Weights
outputs:
    scores                                          ▷ Scores of Classifiers
    classifier_recommended                          ▷ Classifier Recommended

for each instance in stream do
    X = instance.X                                  ▷ Values of Attributes
    y = instance.y                                  ▷ Real Class
    for each classifier in classifiers do
        y' = classifier.test(X)                     ▷ Predicted Class
        classifier.update_error_rate(y, y')
        if classifier.detector.drift is True then
            classifier.reset()          ▷ Resetting the Classifier and its Detector
        else
            classifier.train(X)
        end if
    end for
    scores = CalculateScores(classifiers, w_e, w_m, w_r)
    classifier_recommended = GetBestClassifier(scores)
    Printout(scores)                    ▷ Printing out the Scores of Classifiers
    Recommend(classifier_recommended)   ▷ Printing out the Best Classifier
end for

function CalculateScores(classifiers, w_e, w_m, w_r)
    errors, memories, runtimes, scores = []
    for each classifier in classifiers do
        errors.append(classifier.error)
        memories.append(classifier.memory)
        runtimes.append(classifier.runtime)
    end for
    NormalizeList(errors, memory, runtimes)
    for i in range(0, classifiers.length) do
        emr = CalculateEMR(errors[i], memories[i], runtimes[i], w_e, w_m, w_r)
        scores.append(1 - emr)
    end for
    return scores
end function

function CalculateEMR(e, m, r, w_e, w_m, w_r)
    return (w_e · e + w_m · m + w_r · r)/(w_e + w_m + w_r)
end function

function GetBestClassifier(scores)
    return classifiers[scores.index(scores.max())]
end function
```

## 4 Experimental Evaluation

In this section, we describe our experimental evaluations against seven synthetic and five real-world datasets from the data stream mining domain [3, 10, 11 and 17]. The Hoeffding Tree (HT), Naive Bayes (NB), Perceptron (PR), K-NN, and Decision Stump (DS) learning algorithms are available in our framework. In all experiments, the parameters of the Hoeffding Tree are set to $\delta = 10^{-7}$ (the allowed chance of error), $\tau = 0.05$ (the tie) and $n_{min} = 200$ (the minimum number of observation at each leaf), as determined in [19]. The Perceptron's learning rate is set to $\alpha = 0.01$, as determined in [11]. From the preliminary experiments, we found that 5-NN is the most suitable classifier of the K-NN family in terms of error-rate, memory usage and runtime results.

In our current experimental setup, the error-rate weight $w_e$, the memory usage weight $w_m$ and the runtime weight $w_r$ are all set to one in order to avoid any bias. We used the Drift Detect Method (DDM) [3] for drift detection, as it is less subject to false alarm and requires less memory [1, 3, and 6]. We report the error-rate, the memory usage (in kilobytes) and the runtime (in seconds) of the classifiers at every thousand instances (this value was set by inspection). All experiments were executed on an Intel Core i7-3770 CPU Quad Core 3.4 GHz with 16 GB of RAM running on Windows 10.

### 4.1 Experiments on Synthetic Datasets

We generated seven synthetic datasets, as described in [3, 17], for our experiments. Each dataset contains two class labels of *positive* and *negative*, and 100'000 (one hundred thousand) instances. Concept drifts occur at every 20'000 instances in the SINE1, SINE2 and MIXED, at every 25'000 instances in the CIRCLES, and at every 33'333 instances in the STAGGER datasets.

- SINE1 · *with abrupt concept drift*: It consists of two attributes $x$ and $y$ uniformly distributed within the interval $[0, 1]$. Instances under the curve $sin(x)$, i.e. the classification function, are classified as positive and other instances as negative. At a drift point, the classification paradigm is reversed, e.g. instances under the curve are classified as negative while the remaining instances are classified as positive.
- SINE2 · *with abrupt concept drift*: Like SINE1, it has two attributes $x$ and $y$ which are uniformly distributed in between 0 and 1. The classification function is $0.5 + 0.3 * sin(3\pi x)$. Instances under the curve are classified as positive while the other instances are classified as negative. At a drift point, the classification paradigm is inverted.
- CIRCLES · *with gradual concept drift*: It involves two attributes $x$ and $y$ which are uniformly distributed. The classification is performed with four disk functions. The boundary of each disk is defined by a circle $<(x_c, y_c), r_c>$ that is further described by the function $(x-x_c)^2+(y-y_c)^2-r_c^2 = 0$, where $(x_c, y_c)$ is the center and $r_c$ is the radius. The parameters for the four disks are $<(0.2, 0.5), 0.15>$, $<(0.4, 0.5), 0.2>$, $<(0.6, 0.5), 0.25>$ and $<(0.8, 0.5), 0.3>$ respectively. A change in the classification function results in a drift.

- STAGGER · *with abrupt concept drift*: It contains only three nominal attributes of *size* {*small, medium, large*}, *color* {*red, green*} and *shape* {*circular, non-circular*}. Before the first drift point, instances are labelled positive if $color = red \wedge size = small$. Then after this point and before the second drift, instances are classified positive if $color = green \vee shape = circular$, and finally after this second drift point, instances are classified positive only if $size = medium \vee size = large$.
- MIXED · *with abrupt concept drift*: It has two Boolean attributes $v$ and $w$, and two numeric attributes uniformly distributed in the interval [0, 1]. The instances are classified as positive, if at least two of the three following conditions are satisfied, namely two of $v$, $w$ and/or $y < 0.5 + 0.3 * sin(3\pi x)$ hold. The classification paradigm is reversed, after each drift point.
- ${\text{SINE1}_1}^{100'000} \bullet {\text{SINE2}_1}^{100'000}$ (S1•S2): We generated this stream by creating the union of SINE1 and SINE2.
- ${\text{STAGGER}_1}^{100'000} \bullet {\text{MIXED}_1}^{100'000} \bullet {\text{SINE1}_1}^{100'000}$ (S•M•S1): We generated this stream through concatenating all the instances in STAGGER, MIXED, and SINE1. That is, we appended all attributes and all classes.

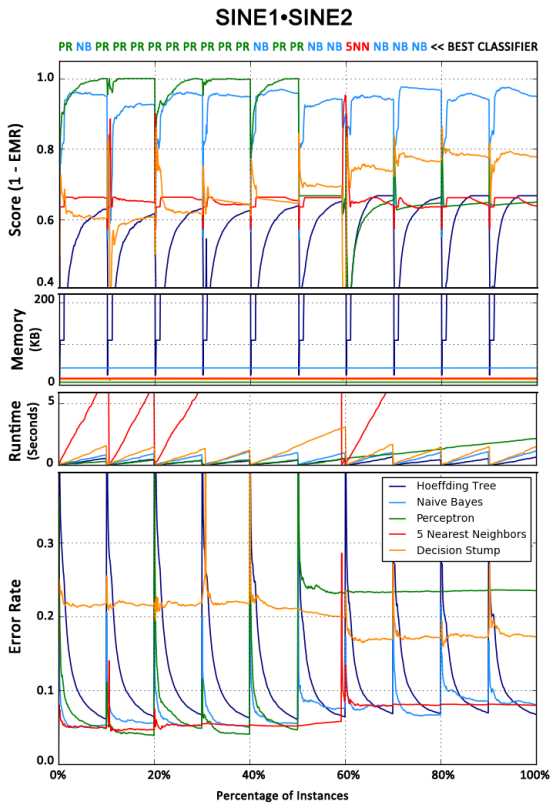



Fig. 4: S1•S2 Data Stream: Scores, Error-rates, Memory Usages and Runtimes

Next, we discuss our experimental results when applying our multi-strategy learning framework to these data streams. The reader should recall that the main aim of these experiments was to illustrate how the "best" type of classifier changes over time, in the presence of concept drift.

We illustrated the evolution of classifiers' scores, error-rates, memory usage and runtime over time for the SINE1•SINE2 and STAGGER•MIXED•SINE1 data streams in Fig. 4 and Fig. 5, respectively. The "current best" classifier is shown at the top of the Figures.

Fig. 4 shows that Perceptron initially performs better than the other classifiers in terms of the combined error-rate, memory usage, and runtime. However, Naive Bayes, Decision Stump and 5-NN are better choices towards the end of the stream. Specifically, Naive Bayes takes over as the best classifier until the end of the stream is reached. Despite the fact that the Hoeffding Tree and 5-NN classifiers have a lower error-rate than Decision Stump, their global performances are lower because of their memory usages and runtimes, respectively.

In Fig. 5, we compare the performances of classifiers as instances arrive over time for S•M•S1. At first, Perceptron globally dominates the other classifiers for the first 20% of the instances. Then, it is outperformed by Decision Stump until 35% of the instances have been observed and to be, in turn, outdone by Naive Bayes for the remainder of the stream, with Perceptron a close second.

We present the average scores for the five types of classifiers in Table 1. These values are included to depict the general behaviours of the classifiers against our synthetic data streams. Our analysis shows that there is no clear winner, in terms of overall performance. However, we notice that the Perceptron and Naive Bayes classifiers are the best at balancing the three indicators. Further, memory usages of Hoeffding trees are prohibitive, while the average runtimes of K-NN are the highest. Thus, the Hoeffding tree algorithm would be better suited in an environment where memory is not a critical resource, while K-NN seems to be unsuitable for building just-in-time models.

Table 1: Average Scores. of Classifiers for the Synthetic Datasets

| DATASETS | HT | NB | PR | 5-NN | DS |
|---|---|---|---|---|---|
| SINE1 | 0.537 | 0.938 | **0.980** | 0.655 | 0.639 |
| SINE2 | 0.595 | **0.951** | 0.665 | 0.649 | 0.786 |
| CIRCLES | 0.517 | **0.918** | 0.663 | 0.635 | 0.739 |
| STAGGER | 0.613 | 0.879 | **0.997** | 0.621 | 0.601 |
| MIXED | 0.569 | **0.940** | 0.869 | 0.635 | 0.648 |
| S1•S2 | 0.567 | **0.932** | 0.809 | 0.656 | 0.692 |
| S•M•S1 | 0.566 | **0.948** | 0.892 | 0.441 | 0.714 |

In this subsection we conducted experiments on the synthetic datasets and we confirmed, as expected, that the "best" classifier changes over the time, as drifts occur. In the next subsection we continue our experiments over real-world datasets.

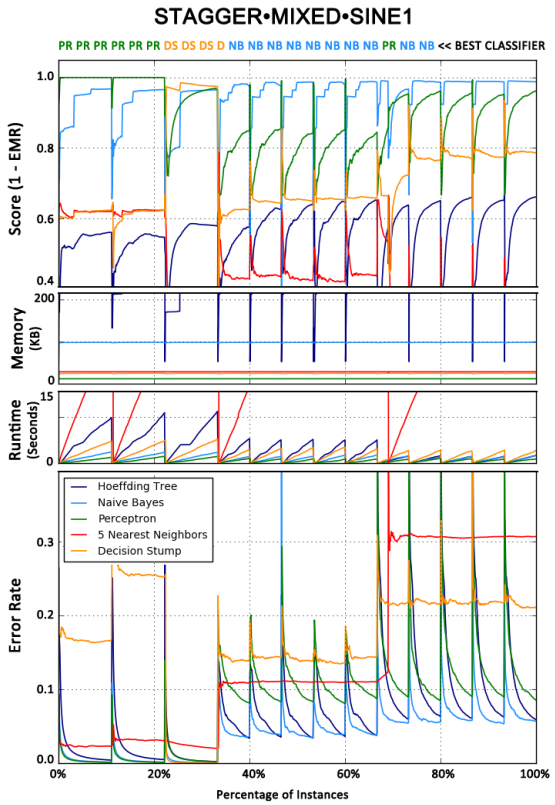



Fig. 5: S•M•S1 Data Stream: Scores, Error-rates, Memory Usages and Runtimes

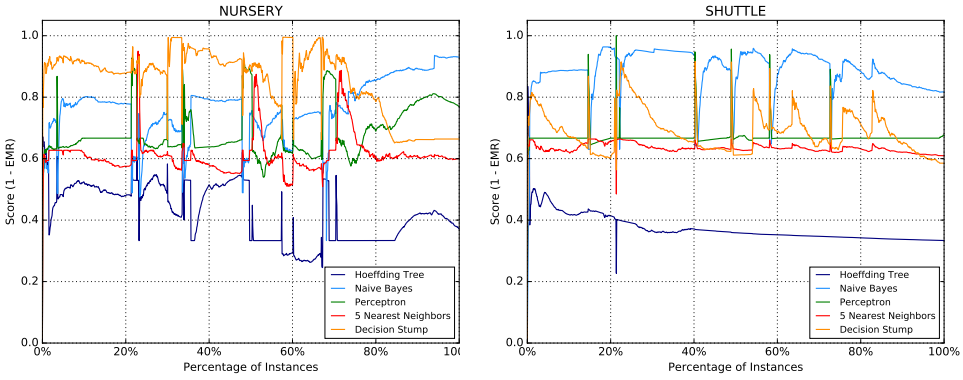
### 4.2 Experiments on Real-World Datasets

In this section, we first describe the real-world datasets used in our experiments. These datasets were obtained from the UCI Machine Learning Repository [20]:

- Nursery [21]: It consists of eight nominal attributes, five class labels and 12'960 instances. This dataset was used to recommend nursery to families.
- Shuttle dataset has nine numeric attributes, seven class labels and 58'000 instances. It was designed to predict suspicious states during a NASA shuttle mission. We use the training set which consists of 43'500 instances.
- Electricity (Elec) [22]: This dataset was collected from the Australian New South Wales Electricity company and has 45'312 instances with one nominal attribute, seven numeric attributes, and two class labels.
- PokerHand [23]: The dataset consists of 1'000'000 instances and eleven attributes. Each record of the PokerHand dataset is an example of a hand consisting of five playing cards drawn from a standard card deck of 52. Each card is described using two attributes (the suit and rank), for ten predictive attributes.

– ADULT [24]: It has six numeric and eight nominal attributes, two classes, and 48'842 instances. It allows predicting if a person has an annual income higher than $50'000.

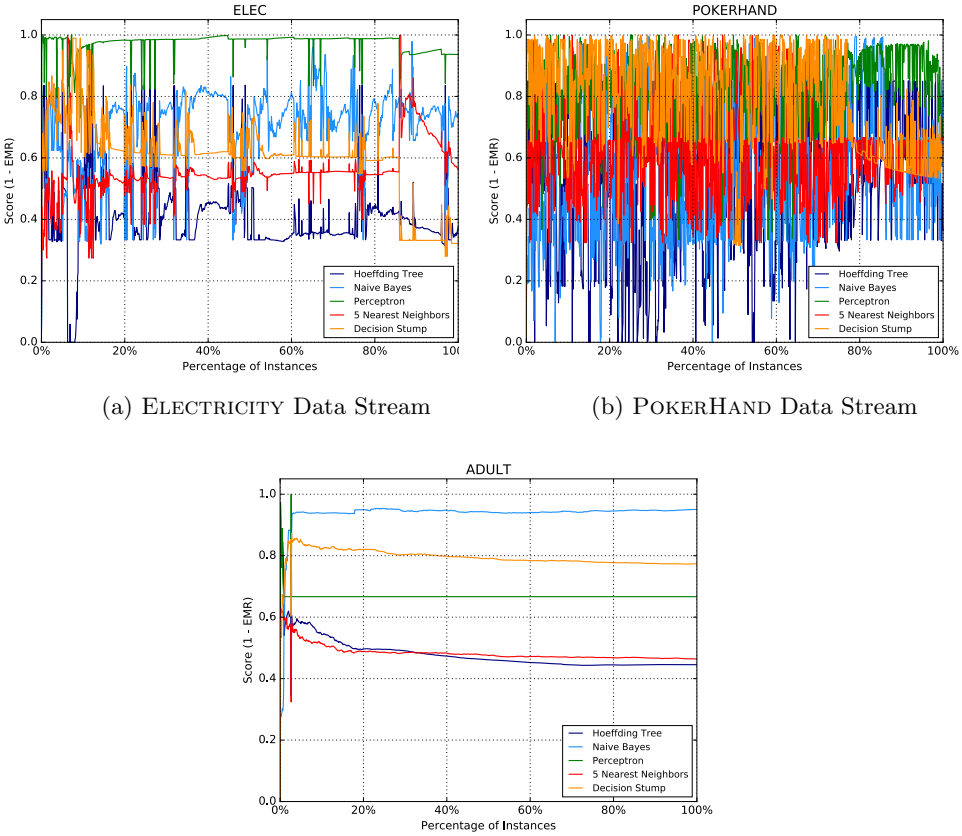
We again followed the *prequential* learning method, as explained in Section 3, and proceed to incrementally construct models against these real-world datasets. Fig. 6 and Fig. 7 show the performances of the various classifiers against the data streams. Note that the "*spikes*" in performance correspond to drift points. These figures clearly illustrate the fact that the performances of the various classifiers considerably vary over time. In order to continuously obtain the best performances, the best classifier must be chosen, leading to an adaptive environment where the models presented to the users change as the data evolve. For example, the initial classifier with the highest score for the NURSERY dataset is Decision Stump, while the Naive Bayes learner performs better towards the end of the stream. For the SHUTTLE data stream, Naive Bayes performs well, except just after concept drift has occurred where Perceptron does best. There is interplay between the Perceptron and Decision Stump classifiers early in the ELECTRICITY data stream, while Naive Bayes and 5-NN also produce high quality models towards the end of the stream. For the POKERHAND, the Perceptron and Decision Stump classifiers are hand-in-hand at the beginning of the stream, while the Perceptron, Hoeffding tree and Naive Bayes classifiers produce the best models later on. The ADULT data stream is the only one that has a clear winner, namely the Naive Bayes learner. Thus, the use of multi-strategy learning does not seem advantageous on this particular dataset.




(a) NURSERY Data Stream (b) SHUTTLE Data Stream

Fig. 6: The Behaviors (overall scores) of Classifiers for NURSERY and SHUTTLE Data Streams

Recall that the aim of our TORNADO framework is to create a multi-strategy environment in which classifiers with diverse learning strategies co-exist, and where the most suitable model is presented to the user at any given time. Our




(a) Electricity Data Stream

(b) PokerHand Data Stream

(c) Adult Data Stream

Fig. 7: The Behaviors (overall scores) of Classifiers for Electricity, PokerHand and Adult Data Streams

system currently includes five different learning algorithms. In the future, we also plan to extend this framework to include other types of classifiers. We are especially interested in including light-weight versions of algorithms. It further follows that the importance of the various weights will heavily depend on the domain of application. We will investigate the optimization of the various weights associated with the measures. Our future work will also focus on dynamically adapting the membership based on the performance of the individual classifiers. We envisage a system where new types of classifiers are added while poorer performing classifiers (e.g. ones that performed well on older data, or never performed well) retire. The valuable retirees are maintained, offline, in order to handle re-occurring concept drifts. Significantly, this will enable the system to revisit history and to adapt to new drifts that follow past patterns.

## 5 Conclusion and Future Work

This paper presented the multi-strategy TORNADO framework in which various diverse learners co-exist. These learners access the same stream and construct their models in *parallel*. In our environment, the current model created by the current "best" type of classifier, at a specific point in time, is presented to the user. We utilized an EMR measure which combines the classifiers' *error-rate*, *memory usage* and *runtime* for performance evaluation. Our experimental results show how the classifier with the highest performance seamlessly adapts, as the stream ebbs and flows.

As stated above, our future research will include varying these three different weights based on domain-specific constraints. When aiming to run classifiers on small devices as the memory restrictions need further consideration. The use of anytime algorithms that are able to fade classifiers in and out (e.g., retiring current poor performers) presents an important new direction [13, 14]. The reader will notice we did not include the total costs of memory in our experiments. We plan to include this value in our decision making. Finally, our framework currently includes five base learners and we will also include other types of learners.

## References

1. Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A Survey on Concept Drift Adaptation. J. ACM Computing Surveys **46**(4) (2014)
2. Hulten, G., Spencer, L., Domingos, P.: Mining Time-Changing Data Streams. In: 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2001)
3. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with Drift Detection. In: 17th Brazilian Symposium on Artificial Intelligence, pp. 286-95 (2004)
4. Gama, J., Fernandes, R., Rocha, R.: Decision Trees for Mining Data Streams. J. Intelligent Data Analysis, **10**(1), pp. 23-45 (2006)
5. Bifet, A., Gavalda, R.: Learning from Time-Changing Data with Adaptive Windowing. In: SIAM International Conference on Data Mining, pp. 443-448 (2007)
6. Huang, D. T. J., Koh, Y. S., Dobbie, G., Bifet, A.: Drift Detection Using Stream Volatility. In: Machine Learning and Knowledge Discovery in Databases, pp. 417-432, Springer Internation Publishing (2015)
7. Koren, Y.: Collaborative Filtering with Temporal Dynamics. In: 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 447-456 (2009)
8. Lee, W., Stolfo, S. J., Mok, K. W.: Adaptive Intrusion Detection: A Data Mining Approach. J. of Artificial Intelligence Review, **14**(6), pp. 533-567 (2000)
9. Stavens, D., Hoffmann, G., Thrun, S.: Online Speed Adaptation using Supervised Learning for High-Speed, Off-road Autonomous Driving. In: 20th International Joint Conference on Artificial Intelligence, pp. 2218-2224 (2007)
10. Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., Gavalda, R.: New Ensemble Methods for Evolving Data Streams. In: 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 139-148 (2009)

11. Bifet, A., Holmes, G., Pfahringer, B., Frank, E.: Fast Perceptron Decision Tree Learning from Evolving Data Streams. In: Advances in Knowledge Discovery and Data Mining, pp. 299-310, Springer Berlin Heidelberg (2010)
12. Zliobaite, I., Budka, M., Stahl, F.: Towards Cost-Sensitive Adaptation: When is it worth Updating Your Predictive Model? Neurocomputing **150**, pp. 240-249 (2015)
13. Olorunnimbe, M. K., Viktor, H. L., Paquet, E.: Intelligent Adaptive Ensembles for Data Stream Mining: A High Return on Investment Approach. In: International Workshop on New Frontiers in Mining Complex Patterns, pp. 61–75, Springer International Publishing (2015)
14. Gaber, M., Stahl, F., Gomes, J. B.: Pocket Data Mining: Big Data on Small Devices. Studies in Big Data 2, Springer Verlag (2014)
15. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive Online Analysis. J. Machine Learning Research **11**, pp. 1601-1604 (2010)
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I. H.: The Weka Data Mining Software: An Update. ACM SIGKDD Explorations Newsletter, **11**(1), pp. 10-18 (2009)
17. Kubat, M., Widmer, G.: Adapting to Drift in Continous Domain. In: 8th European Conference on Machine Learning, pp. 307-310, Springer Verlag, (1995)
18. Gama, J., Sebastiao, R. Rodrigues, P. P.: On Evaluating Stream Learning Algorithms. J. Machine Learning **90**(3), pp. 317-346 (2013)
19. Domingos, P., Hulten, G.: Mining High-Speed Data Streams, In: 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 71-80 (2000)
20. Lichman., M.: UCI Machine Learning Repository. University of California Irvine, School of Information and Computer Science (2013)
21. Zupan, B., Bohanec, M., Bratko, I., Demsar, J.: Machine Learning by Function Decomposition. in: International Conference on Machine Learning (ICML), pp. 421-429 (1997)
22. Harries, M.: Splice-2 Comparative Evaluation: Electricity Pricing. Technical Report, University of New South Wales, Australia (1999)
23. Cattral, R., Oppacher, F., Deugo, D.: Evolutionary Data Mining with Automatic Rule Generalization. Recent Advances in Computers, Computing and Communications, pp. 296–300 (2002)
24. Kohavi, P.: Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid. In: 2nd Int. Conference on Knowledge Discovery and Data Mining (1996)