Adaptive and Reinforcement Learning Approaches for Online Network Monitoring and Analysis

Sarah Wassermann, Thibaut Cuvelier, Pavol Mulinka, Pedro Casas

Adaptive and Reinforcement Learning Approaches for Online Network Monitoring and Analysis

Sarah Wassermann, Thibaut Cuvelier, Pavol Mulinka, Pedro Casas

***Abstract*—Network-monitoring data commonly arrives in the form of fast and changing data streams. Continuous and dynamic learning is an effective learning strategy when dealing with such data, where concept drifts constantly occur. We propose different stream-based, adaptive learning approaches to analyze network-traffic streams on the fly. We address two major challenges associated to stream-based machine learning and online network monitoring: (*i*) how to dynamically learn from and adapt to non-stationary data changing over time, and (*ii*) how to deal with the limited availability of labeled data to continuously tune a supervised-learning model. We introduce ADAM & RAL, two stream-based machine-learning techniques to tackle these challenges. ADAM relies on adaptive memory strategies to dynamically tune stream-based learning models to changes in the input data distribution. RAL combines reinforcement learning with stream-based active-learning to reduce the amount of labeled data needed for continual learning, dynamically deciding on the most informative samples to learn from. We apply ADAM & RAL to the real-time detection of network attacks in Internet network traffic, and show that it is possible to continuously achieve high detection accuracy even under the occurrence of concept drifts, limiting the amount of labeled data needed for learning.**

***Index Terms*—Stream-based Machine Learning; Active Learning; Reinforcement Learning; ADWIN; Network Attacks; MAWILab.**

I. Introduction

NETWORK-traffic monitoring and analysis is paramount to understand the functioning of complex large-scale networks, especially to get a broader and clearer visibility of unexpected events. One of the major challenges faced by online network-monitoring applications is the processing and analysis of large amounts of heterogeneous and fast incoming network-monitoring data. These data usually come in the form of high-speed streams, which need to be rapidly and continuously processed. In this context, detecting and adapting to strong variations in the underlying statistical properties of the modeled data makes data-stream analysis a challenging task.

The application of machine-learning models to network-security and anomaly-detection problems has largely increased in the last decade. However, the general approach in the literature still considers the analysis as an offline learning problem, where models are trained once and then applied to the incoming measurements. This approach is very restrictive when dealing with highly dynamic environments, where concept drifts – i.e., changes in the underlying properties of the prediction target – occur often and previous knowledge rapidly becomes obsolete. Nevertheless, Li et al. demonstrate the advantages and feasibility of online machine-learning applied in industrial control systems to detect cyber-attacks in real-time [1]. An additional challenge of learning in in-the-wild networking scenarios is the lack or limited availability of ground truth or labeled data for training purposes. Labeling new incoming data is often an expensive and cumbersome process – especially when done manually and in an online fashion. Furthermore, not all data samples are equally valuable.

In this paper, we investigate stream-based approaches applied to machine-learning-based network security, using different algorithms for the analysis of continuously evolving data. Stream-based machine-learning analysis consists of processing one data instance at a time, inspecting it only once, and as such, using a limited amount of memory; stream approaches work in a limited amount of time, and have the advantage to perform predictions at any point in time during the stream.

We introduce novel stream-based, continuous learning strategies to deal with the aforementioned challenges. We conceive, describe, and evaluate ADAM & RAL, two stream-based machine-learning approaches to deal with (*i*) concept drifts in the stream of network measurements and (*ii*) limited availability of labeled, ground-truth measurements. ADAM relies on simple data-distribution change-detection algorithms to dynamically adapt the *learning memory* of different stream-based machine-learning models to the most recent data distribution, triggering new learning steps when concept drifts are detected. In the context of stream-based learning, the concept of *learning memory* refers to the set of past measurements which are kept by the system for the purpose of model (re)training. All additional measurements not belonging to the learning memory are discarded. In its simplest form, such a learning memory is implemented as a sliding window of fixed length T_m, which keeps the last m monitored measurements available for further analysis. Naturally, the size m of the learning memory plays a key role, especially when the environment evolves fast.

RAL consists of a stream-based active-learning strategy to reduce the amount of labeled data needed for learning, dynamically deciding on the most *informative* measurements to integrate into the continuous learning scheme. The term *informative* refers here to the novelty of the corresponding measurement, as well as to its ability to improve the prediction performance of the underlying learning model. For example, if the system has already seen a very similar measurement to the one under analysis, or if the model has already a high prediction performance for the region of the input space where the new measurement belongs to, then it is probably

S. Wassermann and P. Casas are with the AIT Austrian Institute of Technology, Austria, T. Cuvelier is with CentraleSupélec, France, P. Mulinka is with CTU Czech Technical University in Prague. Contact: sarah@wassermann.lu, pedro.casas@ait.ac.at

useless to use this new measurement for learning purposes, as there would not be any gains from it. This is the rationale behind active learning. Active learning aims at labeling only the most informative samples to reduce the overall training cost. There is a long list of data-querying strategies and algorithms to decide which data samples should be labeled [2]; among them, the most popular strategy is based on uncertainty sampling, which uses the model-prediction uncertainty for the corresponding sample to decide whether to query its label or not. The higher the uncertainty of the model for a given sample, the more interesting the label of this sample becomes for adapting the model. RAL improves model training and prediction performance by additionally learning from the relevance of its previous sample-selection decisions, using a reinforcement-learning scheme.

We evaluate the performance of the proposed approaches on the detection of different types of network attacks and anomalies, using real network measurements collected at the WIDE backbone network, relying on the well-known MAWILab dataset for attack labeling [3]. Results not only show that particular stream-based machine-learning models are able to keep up with important concept drifts in the underlying network data streams while keeping high detection accuracy, but also that it is possible to drastically reduce the amount of labeled data with stream-based active-learning approaches by relying on reinforcement-learning principles. As an additional contribution to the community, we make RAL freely available on GitHub as a Python package[1].

This paper builds on and extends our recent work on adaptive learning for network monitoring [4], in multiple directions. In particular: (*i*) it brings a more comprehensive overview on the state of the art in adaptive learning; (*ii*) it provides an extended evaluation of ADAM for different types of attacks, as well as a comparative analysis against other adaptation strategies, besides evaluating non-adaptive learning approaches; (*iii*) it develops a theoretical analysis on the expected performance of RAL, in particular with respect to the implemented reinforcement-learning policy; (*iv*) it further evaluates RAL in other datasets, to show the general advantages of the proposal; (*v*) last but not least, it presents (and evaluates) the integration of both ADAM and RAL into a single, reinforcement-based, adaptive active-learning system, adding explicit concept-drift detection into RAL.

The remainder of this paper is structured as follows. Section II presents an overview on the related work. Sections III and IV introduce the proposed ADAM and RAL approaches. In the specific case of RAL, we additionally provide a theoretical analysis of the expected performance to be achieved by the reinforcement-learning loop. Section V details evaluation results on the continuous detection of network attacks and anomalies on real network measurements, using both ADAM and RAL with different machine-learning models. For the sake of completeness, evaluations also consider the analysis of other datasets not linked to network security, as well as the integration of both ADAM and RAL approaches within a single system. Finally, Section VI concludes the paper.

[1]https://github.com/SAWassermann/RAL

II. State of the Art

The application of machine learning to networking problems has been largely explored in the literature [5]. There are a couple of extensive surveys on any-domain anomaly-detection techniques [6] as well as on network-oriented anomaly detection [7], [8], including machine-learning-based approaches. We refer the interested reader to [5] for a detailed survey on the different machine-learning techniques commonly applied to network-traffic analysis.

We have been recently working on the application of machine-learning models to network anomaly-detection problems [9], [10], benchmarking the performance of standard machine-learning models for network anomaly detection [9], further studying more complex and robust models based on ensemble machine-learning techniques [10]. The main limitation of these approaches as compared to our proposals here is that they consider the offline analysis of network measurements, in batch mode.

The specific application of stream-based machine-learning approaches to network security and anomaly detection is by far more limited; a relevant and representative example linked to current research is presented in [11], where Carela et al. evaluate stream-based traffic-classification approaches based on Hoeffding adaptive trees [12], using MAWILab data and the MOA machine-learning toolkit, as we do in this work.

Naturally, the data-stream machine-learning domain has a long-standing tradition and many interesting references are worth mentioning when considering the application and evaluation of stream-based machine-learning models; these cover general problems related to the learning properties for stream-based algorithms [13], [14], the mining and evaluation processes when dealing with massive datasets [15], the identification of model-evaluation issues [16], as well as propositions of general frameworks for data streaming [17]. Recent work presents online-learning strategies dealing with potentially problematic environments, i.e., environments that might not represent the true context – for instance, a person using an account who is not the usual one logging in with these credentials [18].

Of particular relevance for stream-based machine-learning-model evaluation are the problems of *class imbalance* and *concept drift*, which are extensively studied in [19]. The problem of concept-drift detection has been largely addressed in the literature. A very popular change detector in the stream-learning domain is known as ADWIN (ADaptive WINdowing), a dynamically adjusting window-based approach introduced in [20]. In [21], Bifet et al. describe a stream-based bagging algorithm using ADWIN for performance monitoring to decide when to retrain a new model. Similarly, Bifet and Gavaldà designed the Hoeffding adaptive tree (HAT) [22] relying on ADWIN to monitor the prediction performance of different branches, and to replace them with new ones in case those are more accurate. Probabilistic Approximate Window (PAW) [23] is another approach to adapt the training-sample window: at every incoming data point, each element in the window gets dropped with an equal probability. In [24], Gama et al. present Drift Detection Method (DDM), a concept-drift

detector which monitors the error rates of machine-learning models. They rely on the assumption that the error rate decreases over time as long as the samples of the stream are drawn from a stationary distribution and thus issues a drift alert as soon as the error rate increases significantly. An extension to DDM, Early Drift Detection Method (EDDM), has been proposed later in [25]; instead of looking only at the error rates themselves, EDDM considers the distances (i.e., the number of samples) between classification errors, which allows EDDM to work well when facing gradual concept drifts. In [26], Losing et al. present Self Adjusting Memory (SAM), a technique using long-term and short-term memory models (LTM and STM), as well as ensembles, to detect concept drifts. SAM trains some models of the ensemble on LTM and others on STM, combining them to maximize accuracy. In general and in most of these models, the model accuracy is widely used to detect drifts. Another example is FLORA2 [27], in which Widmer et al. use two sliding windows, one for training the model, the other one to compute its accuracy; as soon as the accuracy drops significantly, FLORA2 detects a drift.

When it comes to active learning, there is a vast literature in the field. For example, Žliobaitė et al. present in [28], [29] three simple approaches for this learning paradigm. Their proposed Randomized Variable Uncertainty approach tackles the problem of stream-based active learning, using the model's prediction uncertainty to decide whether to query and tries to overcome concept drifts by randomizing the certainty threshold used for labeling decisions. In [30], Xu et al. develop an active-learning algorithm with two different classifiers: one "reactive" and one "stable". The stable classifier is trained on all available labeled instances, while the reactive one is trained based on a window of recent instances. In [31], Ienco et al. present an active-learning technique based on clustering and prediction uncertainty. In [32], Krawczyk et al. conceive an approach relying on a modification of the Naïve Bayes classifier to update the different learners through the queried samples. In particular, they use one-versus-one classifiers to tackle multiclass problems and update the weights of the different classifiers by comparing their predictions to the ground truth. Their technique behaves similarly to RAL. However, the major difference is that [32] uses information about the classifiers' prediction certainty (without considering the corresponding weights) to adapt the minimum threshold for querying the oracle, while we rely on the usefulness of the decisions taken by RAL to tune the system according to the data stream. Sinha et al. [33] propose to use adversarial machine-learning models: if the model cannot correctly infer that the considered unlabeled sample has not yet been labeled, then its label is unlikely to improve the model performance.

Finally, ideas from reinforcement learning have already infused into the active-learning domain, but mostly into pool-based approaches. In [34], [35], authors rely on the multi-armed bandit paradigm. In [35], Hsu et al. develop ALBL, which uses a modified version of EXP4 [36], a weight-updating rule, to attribute adaptive weights to different learners based on rewards; the learner to use is then determined through these weights and uses its uncertainty measure to select the samples in the pool to hand to the oracle. The approach described in [34] is similar to the one in [35], except for the reward-computation scheme. The algorithm presented in [37], [38] relies on the same principles as the approach we are proposing, but tackles a different problem: Song's goal is to introduce an active-learning component into a contextual-bandit problem, while we are aiming at solving an active-learning problem by using contextual bandits.

Other recent papers dealing with active learning and reinforcement learning include [39]–[42]. However, most of them consider only one of the perspectives addressed by RAL, namely the enhancement of pool-based active learning through reinforcement learning [40]–[42], or the application of active learning to the streaming setup [39]. Combining active learning with reinforcement learning in a streaming, adaptive learning context is the most important contribution of RAL, a very timely yet vaguely addressed problem in the literature. Last, the idea of *learning to active learn*, i.e., data-driven active learning, is developed in [43], [44]. In particular, Konyushkova et al. [43] propose this view on pool-based active learning: the querying decision for a sample is based on an estimation of the accuracy improvement. In [44], Woodward et al. use reinforcement learning in stream-based active one-shot learning, but this work is different from RAL on multiple aspects: (*i*) it tackles a different learning task, as it aims at detecting new classes instead of improving overall classification accuracy, (*ii*) their scheme relies on reinforcement learning only during the training phase and is not updated anymore once deployed, while RAL continuously adapts its querying policy through this learning paradigm during the whole incoming stream, and (*iii*) the system heavily relies on deep recurrent networks, too cumbersome to use in real-time resource-constrained scenarios, unlike RAL. Huang et al. [45] extend this work in several aspects, for instance, reducing the number of queries and increasing the convergence speed by adding a cross-entropy term in the loss function when training the underlying neural network. Puzanov et al. introduce DeROL [46], a one-shot-learning framework based on active learning and deep reinforcement learning designed to optimize labeling-resource allocation in expert-based systems.

III. ADAM – Learning with Adaptive Memory

We start by introducing ADAM, relying on an ADAptive Memory strategy. Given that we are dealing with continuous data analysis, the approach must be able to identify and adjust to the variation of the statistical properties of the analyzed data, detecting sudden statistical changes, namely concept drifts. To do so, ADAM relies on ADWIN, an approach which maintains a window of variable size containing training samples. The algorithm automatically grows the window when no change is apparent, and shrinks it when the statistical properties of the stream change. ADWIN automatically adjusts its window size to the optimum balance point between reaction time and small variance. ADAM uses ADWIN on top of four stream-based machine-learning algorithms popular in the literature, including incremental k-NN, Hoeffding adaptive trees (HAT), adaptive random forests (ARF) [47], and SVM through stochastic gradient descent (SGD).

Algorithm 1 ADWIN algorithm.

```
1: procedure ADWIN(ε)
2:   initialize window W
3:   for each n > 0 do
4:     W ← W ∪ {x_n}                ▷ add x_n to the head of W
5:     if |μ̂_{W_0} − μ̂_{W_1}| ≥ ε for some split of W = W_0 · W_1 then
6:       drop instances from the tail of W
```

A. Adaptation Strategy

Stream-based machine-learning models continuously adapt to the changes of the underlying statistics describing the current data under analysis, by periodically retraining. To deal with evolving data, one needs to define strategies to: firstly, detect when changes occur; secondly, decide which data to use for a subsequent model retraining; and last, retraining or recalibrating the model when a significant change has been detected. There are multiple strategies to deal with concept drift in practice. A trivial one is to periodically retrain the learning model with the most recent historical data. Depending on how dynamic the environment or problem under analysis is, this retraining could be done on a monthly, weekly, or even daily basis. When dealing with highly changing environments, such as the case of network security, it seems appropriate to only include a small portion of the most recent historical data in the training set to best capture the new relationships between inputs and outputs. This can be realized by using sliding windows.

The most basic adaptation strategy for periodic retraining consists of using a sliding window containing recent measurements to retrain the model. At time t_0, the model is trained on the most recent data contained in a fixed-length window of size T_m, and used to predict the label of a newly arrived sample. From time t_1 onwards, the sliding window gets new data and triggers a retraining of the model. We refer to this approach as FIXWIN (FIXed-length WINdowing). FIXWIN provides soft adaptations of the learning model, as the memory T_m of the system operates as a smoothing filter, and therefore usually performs well under gradual or incremental drifts. However, when abrupt changes occur, FIXWIN introduces a potentially significant delay in the adaptation process.

A better way to deal with abrupt changes is through reactive strategies. In a nutshell, a reactive learning strategy consists also of a sliding window keeping the most recent historical data for retraining, but additionally adds a change-detection algorithm to rapidly and automatically identify concept drifts, discarding all data belonging to the previous distribution. ADWIN falls into this category. The ADWIN algorithm [20] keeps a sliding window W with the most recently observed measurements $\{x_i, x_{i+1} \dots x_n\}$, where x_i is the first sample included in the window at time n. Without loss of generality, we assume that, at each time n, instance x_n is generated according to an unknown probability distribution D_n. Let μ_n be the (unknown) expected value of x_n. Let m be the length of W, and $\hat{\mu}_W$ the (computed) average of the measurements in W. The idea of ADWIN is straightforward: whenever two *large enough* sub-windows of W exhibit *distinct enough* averages, we conclude that the corresponding expected values are different, and the older portion of the window is dropped. Algorithm 1 briefly describes ADWIN; $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ are the averages of the m_0 and m_1 instances in W_0 and W_1, respectively, and ε is a threshold. It is not needed to define m_0 and m_1, as these are decided by the algorithm itself. ADWIN is basically a statistical test for different distributions in W_0 and W_1, which checks whether the observed average in both sub-windows differs by more than the threshold ε. For the statistical test, the null hypothesis is the absence of change (μ_n remains constant within W).

Let $\delta \in (0, 1)$ be a confidence value defined as input to ADWIN, acting as an upper bound to the false positive rate of the change detection – shrinking W, when μ_n was actually constant within W. According to [20], the threshold ε is defined based on the global error of the statistical test (the false positive rate) and on the lengths m_0 and m_1 of the corresponding sub-windows. More precisely, if we define q as the harmonic mean of m_0 and m_1, and $\delta' = \delta/m$, then:

$$\varepsilon = \sqrt{\frac{1}{2q} \cdot \log \frac{4}{\delta'}}$$

The role of δ' is to avoid problems with multiple hypothesis testing, since ADWIN tests m different possibilities for W_0 and W_1, and we want a global error below δ.

Following previous work on adaptive learning [21], [22], ADAM uses ADWIN to detect drifts in the time series of the model-prediction accuracy. More precisely, the drift detector is fed with the performance of the ensemble model for each sample in the stream (i.e., whether the ensemble provided the right prediction or not). The rationale behind this is straightforward: if the model is performing well, and the accuracy starts dropping, then it makes sense to retrain the model, as it is no longer accurate – meaning that the new data is distributed according to an unseen process. On the contrary, if the model is performing badly, and suddenly starts performing better, then there are high chances of boosting the performance by adapting to current data distribution. As soon as a drift has been detected, ADAM retrains the underlying model using only the most recent window W_1.

B. Concept-Drift Detection

Concept drift happens when the statistical properties of the analyzed dataset abruptly shift in time [48]. Different change-detection algorithms can be applied to identify the times when the probability distribution of a stochastic process or time series changes. In our problem, such a detection has to be performed in an online manner, i.e., without assuming that the statistics of the complete time series are known in advance. The ADWIN algorithm is by design an online change-detection algorithm. In this paper, we consider an additional change-detection algorithm to analyze the considered dataset: the Page-Hinkley test (PHT) [49]. PHT is a standard statistical test for change detection, commonly used in time-series analysis. In a nutshell, this test is a sequential adaptation of a simple change-detection test for an abrupt change of the average of a Gaussian stochastic process, and it allows efficient

detection of changes in the usual behavior of a process. Similar to ADWIN, the null hypothesis corresponds to an absence of change. We refer the reader to [49] for further details on the PHT test.

IV. RAL – Stream Learning with Active Reinforcement

RAL relies on reinforcement-learning principles, using rewards and contextual-bandit algorithms [36], as well as prediction uncertainty. The overall idea is summarized in Figure 1. The intuition behind the different reward values is that we attribute a high (positive) reward in case the system behaves as expected, and a low (negative) one otherwise, to penalize it. RAL obtains rewards/penalties as soon as it is asking for ground truth. In a nutshell, it earns a positive reward ρ^+ in case it queries the oracle and would have predicted the wrong label otherwise (the system made the right decision to ask for the ground truth: the sample is deemed informative) and a penalty ρ^- (a negative reward) when it asks the oracle even though the underlying classification model would have predicted the correct label (querying was unnecessary). The rationale for using reinforcement learning is that RAL learns not only based on the queried samples themselves, but also from the usefulness of its decisions. The objective function to maximize is the total reward: $\sum_{i=1}^{n} r_i$, where r_i is the i-th reward (ρ^+ or ρ^-) obtained by RAL.

The conceived system additionally makes use of the prediction certainty of the underlying classification model(s). The prediction certainty is defined as the highest posterior classification probability among all possible labels for sample x. More formally, the prediction certainty of a model is equal to $\max_{\hat{y}} P(\hat{y}|x)$, with $\hat{y}$ being one of all the possible labels for x. The rationale behind this design choice is that the model's prediction uncertainty is an appropriate proxy for assessing the usefulness of a data point. Combining the reward mechanism with the model's uncertainty allows us to tune the sample-informativeness heuristic to better guide the query decisions.

Also inspired by the bandit literature [50], to better deal with concept drifts in the data, we implement an ε-greedy policy, which improves the data-space exploration; we sample a uniform probability distribution, and if this value is below a certain threshold ε, the system queries the oracle, ignoring the decision of RAL's classification models. We refer to this as the *ε-scenario*. This ensures that we have a good chance of detecting potential concept drifts: without this policy, the system could end up being too confident about its predictions, and thus never ask the oracle again, even though its estimations are wrong.

Next, we present the details of a committee or multiclassifier version of RAL, relying on multiple models. Nevertheless, it is very easy to use RAL with a single machine-learning model. We provide some comments on this by the end of the section.

A. RAL Algorithmic Details

The algorithm behind RAL is summarized in Algorithm 2. Our approach is inspired by contextual bandits [36]. We rely on a set of experts (i.e., different machine-learning models), referred to as a *committee*. Each expert gives its opinion for the sample to consider: should the system ask the oracle for feedback or is the expert confident enough about its prediction? To assess a model's prediction certainty, we rely on a certainty threshold θ: if the model's certainty is below θ, the expert is too uncertain about the prediction to make and it thus advises that RAL asks for the ground truth. The query decision of the committee takes into account the opinions of the experts, but also their decision power: if the weighted majority of the experts votes against querying, RAL will rely on the label prediction provided by the committee, used in the form of a voting classifier. The decision power of each expert gets updated such that the experts which agree with the entire committee are obtaining more power in case that particular decision is rewarding, i.e., informative (otherwise, these experts get penalized). These weights are updated through the EXP4 rule [36], with a learning rate η. RAL does not update the decision powers of the different learners in the ε-scenario: the committee did not take the querying decision and therefore the weights of the models should not be impacted by this querying action. Thus, RAL is a model-free system: it does not build a model of its environment; it primarily relies on learning from its interactions with the data stream as opposed to planning.

The computation of the reward is carried out every time the committee decided to query (i.e., not in the ε-scenario).

Algorithm 2 RAL algorithm.

procedure RAL(x, E, α, θ, ε, η)
  x: sample to treat
  E: set of learners, members of the committee
  α: vector of decision powers of learners in E
  θ: certainty/querying threshold
  ε: threshold for ε-greedy
  η: learning rate
  decisions $\leftarrow$ {} ▷ will contain decisions of learners
  for e $\in$ E **do**
    decisions[e] $\leftarrow$ e.askCertainty(x) $< \theta$
  committeeDecision $\leftarrow$ round($\sum_{e \in E} \alpha[e] \cdot$ decisions[e])
  $p \leftarrow \mathcal{U}_{[0,1]}$ ▷ random number drawn from a uniform distribution
  if $p < \varepsilon$ **or** committeeDecision = 1 **then**
    $y \leftarrow$ acquireLabel(x)
  if committeeDecision = 1 **then**
    $r \leftarrow$ getReward(x, y)
    $\alpha \leftarrow$ updateDecisionPowers(r, E, decisions, committeeDecision, α, η)
    $\theta \leftarrow \min\left\{\theta\left[1+\eta\times\left(1-2^{\frac{r}{\rho^-}}\right)\right], \quad 1\right\}$
function UPDATEDECISIONPOWERS(r, E, decisions, committeeDecision, α, η)
  for e $\in$ E **do**
    if decisions[e] = committeeDecision **then**
      $\alpha[e] \leftarrow \alpha[e] \times \exp(\eta \times r)$ ▷ EXP4
  return $\alpha / \sum_{e \in E} \alpha[e]$ ▷ normalize α
function GETREWARD(x, y)
  return (ρ^- **if** $\hat{y}(x) = y$ **else** ρ^+)

RAL therefore gets rewarded when it queried the oracle and asking was informative (i.e., the voting classifier would have predicted the wrong label). Conversely, RAL is penalized if the system used the oracle because the committee decided to do so, even though the underlying classifier would have predicted the correct class. More formally, the reward function r_n of RAL for its n-th query is the following:

$$r_n = \begin{cases} \rho^+ > 0, & \text{if asking was informative} \\ \rho^- < 0, & \text{if asking was unnecessary} \end{cases}$$

As an additional step, to ensure that RAL adapts in the best possible way to the data stream, we do not only tune the weights of the committee members based on rewards, but also the uncertainty threshold θ, denoted in the remainder of this section as θ_n to stress that it is influenced by the $n-1$ samples observed so far. Again, as for the decision powers, θ_n is not updated in the ε-scenario. The update rule of θ_n we implemented for our tool is written as follows:

$$\theta_n \leftarrow \min\left\{\theta_{n-1} \times \left[1 + \eta \times \left(1 - 2^{\frac{r_n}{\rho^-}}\right)\right], \quad 1\right\}$$

We now detail the reasoning behind the selection of the update policy used by RAL. We are looking for an update rule of the form

$$\theta_n \leftarrow \min\left\{\theta_{n-1} \times [1 + f(r_n)], \quad 1\right\}$$

where $f(r_n) = 1 - \exp(a \times r_n)$. The threshold should increase slightly when the reward is positive, conversely when the reward is negative. More formally, the update policy should satisfy the following properties:

1 – θ_n **should decrease *fast* in case r_n is negative**, as this indicates the system queries too often, thus is doing poorly. Therefore, θ_n should be adapted fast to improve RAL's performance.

2 – θ_n **should *slightly* increase when r_n is positive**, so that the system does not always keep decreasing the threshold and avoids that θ_n drops to 0. The model was right to ask for more samples, and thus the threshold should be increased. Nevertheless, as the system is currently doing well, we do not want the threshold to be too reactive to the queries.

3 – f **must have two extrema**: a minimum at $\rho^- < 0$ and a maximum at $\rho^+ > 0$.

4 – θ_n **represents a probability**. $\theta_n = 0$ is not acceptable due to the product form of the update policy, thus the values of θ_n must be in the interval $(0, 1]$.

5 – $f(r_n)$ **must be in the interval** $(-1, 1]$ to ensure that θ_n takes values corresponding to a probability. We exclude -1 from the allowed range of values to avoid that θ_n drops to 0.

Properties 1 and 2 lead us to choose the family of functions $f : x \mapsto 1 - \exp(a \times x)$ parameterized by a. Property 5 can be translated into an equation to determine this parameter by imposing the range constraint:

$$\lim_{r \to \rho^-} f(r) = 1 - \exp(a \times \rho^-) = -1$$

After solving this equation, we get $a = \frac{\ln 2}{\rho^-}$. As f is strictly increasing, and because a is nonpositive, f will have a maximum when $r_n = \rho^+$ (thus satisfying property 3). Note that, in order to satisfy property 5, ρ^+ must be chosen such that $f(\rho^+) \leq 1$.

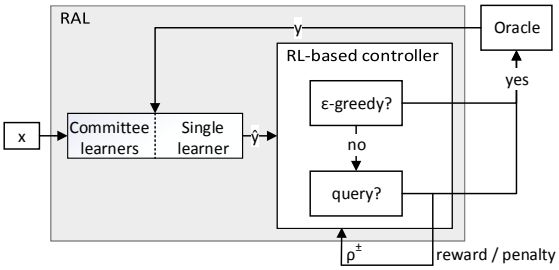


Figure 1: Overall idea of the system.

As a final step, we introduce an additional hyper parameter to the update rule, namely the learning rate η. This rate aims at smoothing the evolution of the threshold θ_n, i.e., avoiding that θ_n changes too dramatically with a single query. We thus have the following update rule:

$$\theta_n \leftarrow \min\left\{\theta_{n-1} \times \left[1 + \eta \times \left(1 - 2^{\frac{r_n}{\rho^-}}\right)\right], \quad 1\right\}$$

We restrict the values of η to the range $(0, 1)$. Indeed, we still must satisfy property 5 (a value of 1 would violate this one) and $\eta = 0$ would lead to a nonreactive system, as the threshold would never adapt.

We acknowledge that RAL includes a non-negligible number of hyperparameters which should be well chosen in order to obtain the best results. While we do not have any rule of thumb on how to define exact values, the following guidelines help RAL learn from the streaming data:

1 – The initial value of θ should be set to a high one (i.e., close to 1) when the number of possible labels is low, to avoid that the model is always too certain about its prediction for the encountered samples.

2 – ε should be higher when dealing with more dynamic datasets, to increase the probability of accurately grasping concept drifts; in general, we would advise using values in the range of 1 to 5%.

3 – η should be small to avoid changing the decision powers of the different learners, i.e., α, and θ_n too abruptly; we would advise values below 0.1.

4 – There is no specific range of values for $\rho^\pm$ which works better than others and these values should be picked considering the situation in which RAL is used: if unnecessary queries are a major issue, one should set ρ^- such that its absolute value is much higher than the one of ρ^+.

To conclude, and as we said before, RAL can also easily be used with a single classifier instead of a committee of learners. Transforming the committee version into a single-classifier one is straightforward. In that case, RAL becomes very lightweight and the only element of the system that allows it to efficiently adapt to and learn from the data stream is the variation of the uncertainty threshold θ, by relying on the rewards.

B. Expected Total Reward Analysis

As the main novelty of RAL lies in the introduction of a reinforcement-learning loop to improve querying effectiveness and the data exploration-exploitation trade-off, we devote this section to the study of the reward properties in RAL. We rely on concepts from the bandit theory to understand its expected behavior. In the general case of a multiclass classification problem, under the assumptions that $\rho^+ \pm \rho^- \geq 0$, we prove the following bounds for the expected total reward of RAL, T being the number of samples in an active-learning session (see Appendix VII for proof details):

$$T\left(\rho^+ - \rho^-\right)\beta \leq \mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\} \leq T\left(\rho^+ + \rho^-\right) + T\left(\rho^+ - \rho^-\right)\beta',$$

where β and β' are functional parameters characterizing the performance of the underlying machine-learning models [51], [52]. For instance, if the models perform poorly, β is close to zero and β' is close to one. Conversely, if the models are excellent, $\beta \approx 1$ and $\beta' \approx 0$. This gives some intuition into the meaning of this result. At the beginning of an active-learning session, models perform poorly, but at least the expected total reward is at least zero.

Furthermore, it is more beneficial to choose the rewards such that $\rho^+ + \rho^- \geq 0$. In this case, the upper bound is higher – we add a term $T(\rho^+ + \rho^-)$ with respect to the scenario where the rewards are not chosen in this manner –, as well as the lower bound – we add a term $T(\rho^+ - \rho^-)$. This means that, for a promising behavior of RAL, good decisions should be more rewarded than bad ones are penalized. At last, results also show that the expected total reward is *significantly higher* than $T\rho^-$, whatever the values of $\rho^{\pm}$: RAL usually takes the appropriate decision, and thus mostly queries when necessary. Conversely, the upper bound is always nonnegative.

V. Continuous Detection of Network Attacks

To evaluate the performance of the proposed algorithms and adaptation strategies, we consider the detection of diverse types of network attacks in real network-traffic measurements collected at the WIDE backbone network, using the well-known MAWILab dataset for attack labeling. MAWILab is a public collection of 15-minute network-traffic traces [3] captured every day on a backbone link between Japan and the US since 2001. Building on this repository, the MAWILab project uses a combination of four traditional anomaly detectors (PCA, KL, Hough, and Gamma) to partially label the collected traffic.

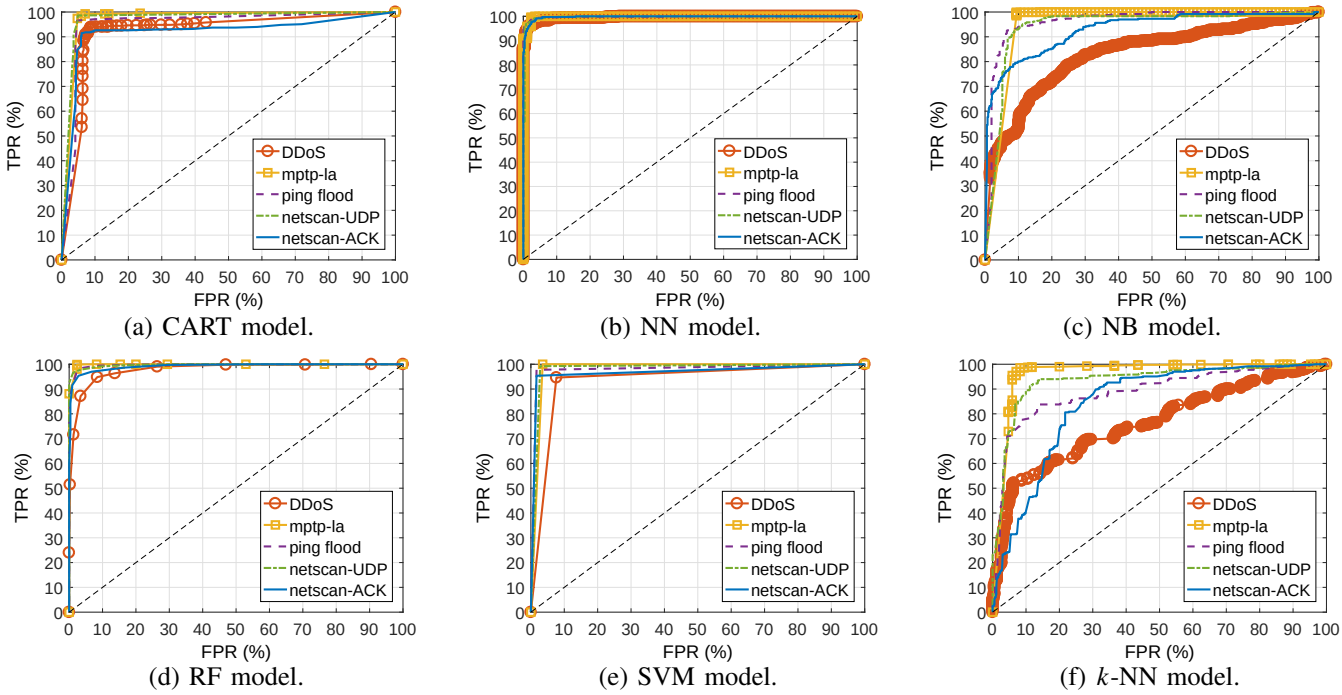
The evaluations provided in this section are broad and comprehensive, covering the overall picture addressed by ADAM and RAL. Given the different nature and goals of ADAM and RAL, we rely on two different evaluation strategies, each of them adapted to the specific challenges tackled. Next, we describe the dataset used in the evaluations in Section V-A, as well as the employed evaluation strategies, along with the obtained results in Sections V-C and V-D. To better understand the properties and advantages of stream-based adaptive learning in practice, we also evaluate the performance of the proposed models in an offline-learning scenario, and show how this approach leads to poor performance when concept drifts occur in Section V-B. Finally, we evaluate a natural extension of ADAM and RAL, by integrating the ADWIN change detector within the RAL approach in Section V-E.

Table I: Input features for detection of attacks.

Field	Feature	Description
Tot. volume	# pkts	num. packets
	# bytes	num. bytes
PKT size	pkt_h	H(PKT)
	pkt_{min,avg,max,std}	min/max/std, $\overline{\text{PKT}}$
	pkt_p{1,2,5,...95,97,99}	percentiles
IP proto	# ip_protocols	num. diff. IP protocols
	ipp_h	H(IPP)
	ipp_{min,avg,max,std}	min/max/std, $\overline{\text{IPP}}$
	ipp_p{1,2,5,...95,97,99}	percentiles
	% icmp/tcp/udp	share of IP protocols
IP TTL	pkt_h	H(TTL)
	ttl_{min,avg,max,std}	min/max/std, $\overline{\text{TTL}}$
	ttl_p{1,2,5,...95,97,99}	percentiles
IPv4/IPv6	% IPv4/IPv6	share of IPv4/IPv6 pkts.
	# IP_src/dst	num. unique IPs
	top_ip_src/dst	most used IPs
TCP/UDP ports	# port_src/dst	num. unique ports
	top_port_src/dst	most used ports
	port_h	H(PORT)
	port_{min,avg,max,std}	min/max/std, $\overline{\text{PORT}}$
	port_p{1,2,5,...95,97,99}	percentiles
TCP flags (byte)	flags_h	H(TCPF)
	flags_{min,avg,max,std}	min/max/std, $\overline{\text{TCPF}}$
	flags_p{1,2,5,...95,97,99}	percentiles
	% SYN/ACK/PSH/...	share of TCP flags
TCP WIN size	win_h	H(WIN)
	win_{min,avg,max,std}	min/max/std, $\overline{\text{TCPF}}$
	win_p{1,2,5,...95,97,99}	percentiles

A. Data Description

The traffic studied in this paper spans two weeks of packet traces collected in late 2015. From the labeled anomalies and attacks, we specifically focus on those which are detected simultaneously by all four MAWILab detectors. We consider five types of attacks/anomalies: (1) DDoS attacks (DDoS), (2) HTTP flashcrowds (mptp-la), (3) flooding attacks (ping flood), and two different flavors of distributed network scans (netscan) using (4) UDP and (5) TCP-ACK probing traffic. We train different models to (binary) detect each of these attack types separately. To perform the analysis in a stream-based manner, we consider a slotted, time-based approach: we split the traffic traces in consecutive time slots of ΔT seconds each, and compute a set of features describing the traffic in each of these slots. In addition, each slot i is assigned a label l_i, consisting of a 5-dimensional binary vector which indicates at each position j if anomaly of type $j = 1..5$ is present or not in the current time slot. We compute a large number of features describing a time slot, using traditional packet-level measurements including traffic throughput, packet sizes, IP addresses and ports, transport protocols, flags, etc. The total set accounts for 245 features, which are computed for every time slot i. Besides using traditional features such as min/avg/max values of some of the input measurements, we also consider the empirical distribution of some of them, sampling it at many



(a) CART model. (b) NB model. (c) NB model.

(d) RF model. (e) SVM model. (f) k-NN model.

Figure 2: Detection performance (ROC curves) achieved by different models for (offline) detection of network attacks.

different percentiles. This provides much richer information, as the complete distribution is taken into account. We also compute the empirical entropy $H(\cdot)$ of these distributions, reflecting the feature dispersion. Table I describes the full set of 245 features.

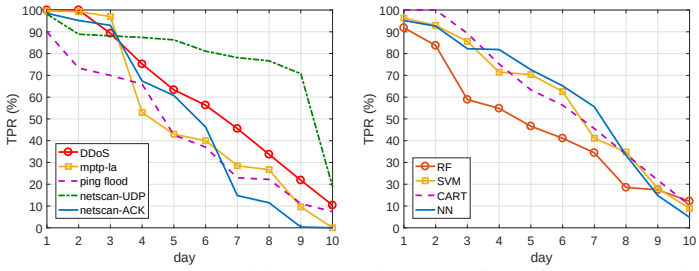
Naturally, the length of a time slot ΔT influences the computation of the proposed features, and therefore the performance of the detection models. While the analysis of such an impact is out of the scope of this paper – the analysis of streaming networking data under the presence of concept drifts, independently of the input features' definition –, we have tried different values for ΔT and adopted $\Delta T = 5$ seconds for the computation of features, which provides a good trade-off between low temporal resolution and model performance.

B. Offline Training and Online Performance Degradation

To understand the limitations of offline-learning approaches and the advantages of stream-based ones, we begin the analysis by treating the detection problem as an offline-training problem. We consider six standard machine-learning models previously used in the literature for the offline analysis, including: (*i*) decision trees (CART), (*ii*) naïve Bayes (NB), (*iii*) multi-layer neural networks (NN), (*iv*) support vector machines (SVM), (*v*) random forest (RF), and (*vi*) nearest neighbors (k-NN). We test the detection capabilities of the six supervised approaches by computing the true and false positive rates (TPR/FPR) for each of the attack types, using as input the full set of 245 features. Figure 2 depicts the receiver-operating-characteristic (ROC) curves obtained with each detector, for the proposed attack classes. To reduce over-fitting, all presented results correspond to 10-fold cross-validation (CV). We use standard Java Weka machine-learning libraries for the analysis, including parameter calibration (grid-based search), model training, and validation. All the attack-detection problems presented next are treated as binary classification tasks; as a consequence, when we refer to the term *detection accuracy*, we shall always refer to the recall of the attack class, and not to the global accuracy of the model.

Figure 2 provides the comparative results obtained for the selected supervised detectors. Besides the NB and the k-NN models, the tested approaches provide all highly accurate results for the five types of attacks. In general, detection performance is worse for DDoS attacks for all the evaluated models, suggesting that its fingerprint in the considered set of features is less marked than for the other attacks. Both the NN and RF models achieve the best performance, detecting around 80% of the attacks without false alarms. The proposed models are very accurate to detect the different types of attacks. The NN and the RF models are the best ones, detecting more than 90% of the attacks with a false alarm rate below 1%. These initial results suggest that offline approaches are in principle highly accurate; however, these results correspond to 10-fold cross-validation, using the full dataset. In practice, one would normally train the models with some time-bounded dataset, and then apply the resulting detectors to the subsequently incoming measurements. To better understand how well it could work in a real deployment, Figure 3 reports the performance of some of the aforementioned models, when trained over the first day of data, and tested on the following days.

Figure 3(a) considers the performance of the CART model – similar results are observed for the other models, for all the five attack types. While detection accuracy remains high during the first couple of days, there is a major performance degradation over time when models are not retrained. The



(a) CART model. (b) Detection of DDoS attacks.

Figure 3: Performance drift for the offline trained models along time. Training is done on the first day of data.

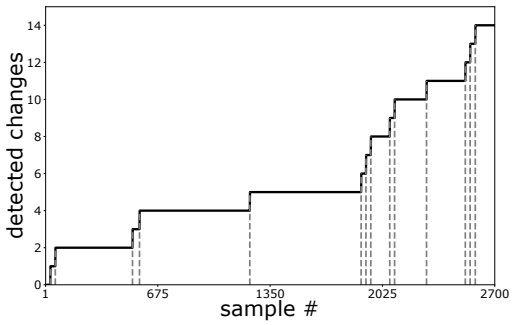


Figure 4: PHT detection; dashed lines indicate changes.

same is observed for other models in Figure 3(b), where detection performance for DDoS attacks is reported – we consider the DDoS attack, as it is the most difficult one to detect in the dataset. This simple example serves as basis to explain the paramount relevance of adaptive learning for network security, particularly in practice, where machine-learning based detectors are generally mistrusted by network operators.

C. ADAM Evaluation Strategy and Performance

We now proceed to the evaluation of the four implemented learning algorithms (incremental k-NN, HAT, ARF, and SGD) and adaptation strategies in ADAM. We firstly study the statistical changes on the input data over the complete evaluation period by detecting the concept drifts. Then, we evaluate the different adaptive learning algorithms using ADWIN, and, for the sake of completeness, complement the analysis when using ADAM with the FIXWIN approach.

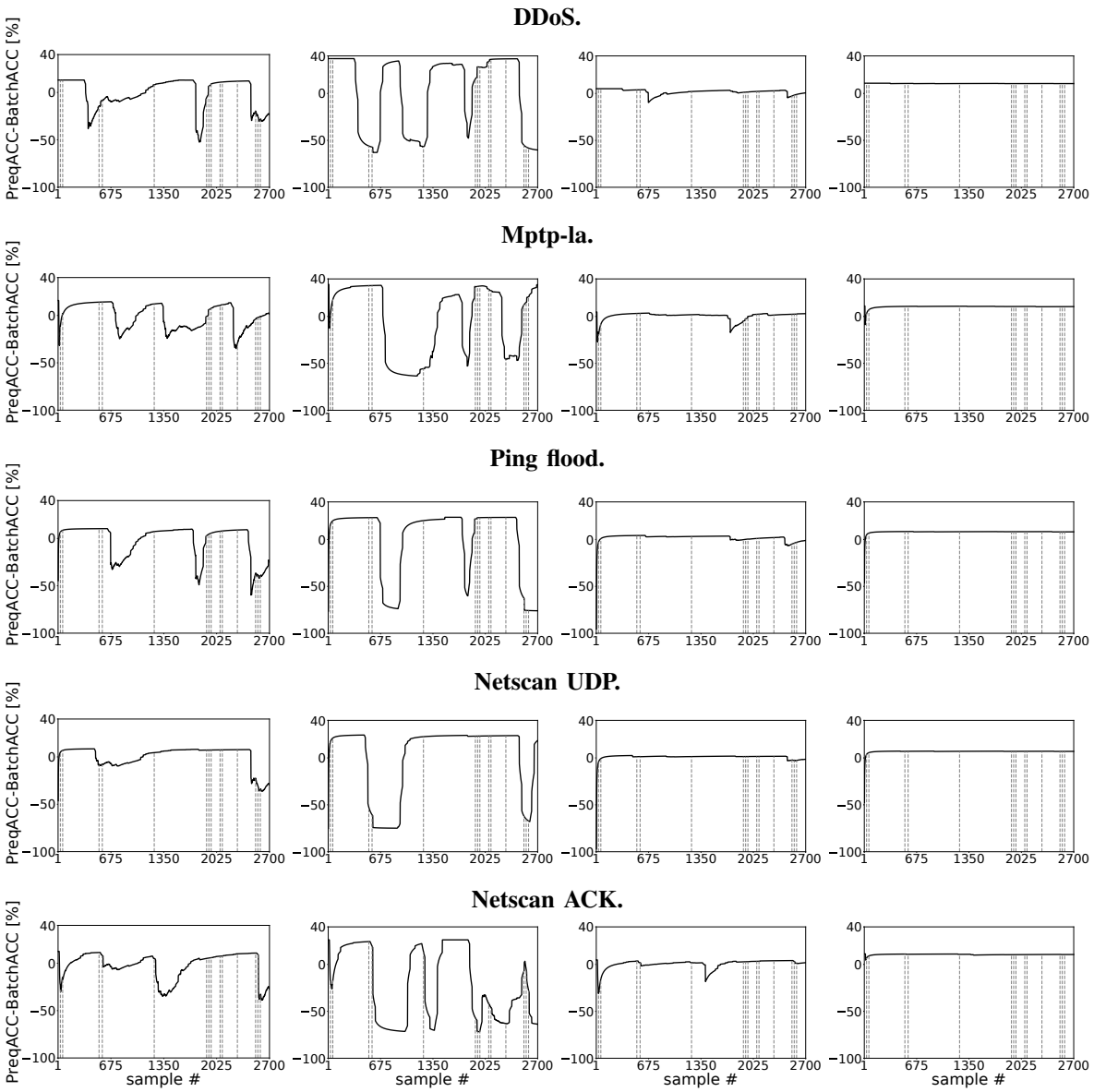
To evaluate the performance of stream-based algorithms, the standard approach in the literature is to benchmark them against their corresponding offline, batch implementations (cf. Figure 2). We use as baseline the results obtained for the matching offline algorithms, including k-NN, decision tree, random forest, and SVM, subtracting the batch results from the corresponding stream-learning results.

A commonly used evaluation scheme in the data-stream-mining domain is the well-known prequential approach. Each instance is first used to test a model, and then to update it. Prequential evaluation can be used to measure the accuracy of a model since the start of the evaluation, by keeping in memory the complete history of instances and evaluating the model on each new instance, but it is generally applied using sliding windows – as we do in ADAM –, which forgets previously seen instances in the model-update process and focuses on those instances in the current sliding window or learning memory. As opposed to more traditional k-fold cross-validation, which is generally used in the evaluation of offline machine-learning models based on k shuffles of the complete dataset, prequential cross-validation works on a single stream of data using only one model: consequently, its assessment of the stream-based model tends to be weaker.

To avoid this weakness, we evaluate ADAM following a new strategy to evaluate stream-based algorithms [53], using prequential k-fold cross-validation. This strategy is basically an adaptation of k-fold cross-validation to the streaming setting, and assumes we have k different models derived from the algorithm we want to evaluate, running in parallel. Each time a new sample arrives, it is used for testing one of the k models selected randomly, and is then used for training by all the other models. As evaluation metric, we take the attack-detection accuracy (ACC) – i.e., the recall for the attack class.

For completeness, we also consider the area under the ROC curve (AUC) as evaluation metric, on its prequential version. The machine-learning community often uses the AUC statistic for model comparison, which is simple and informative, and provides more reliable comparisons when dealing with imbalanced data. To explain the AUC, let us consider a 2-class classification model f that first estimates a probability of a sample x belonging to the class +, denoted by $f(x)$, then uses this probability to make a decision. If x^+ is a random sample of the class + and x^- another random sample of the class −, the AUC is the probability that $f(x^+) > f(x^-)$. The higher the AUC, the better the discrimination of the model. The AUC also reveals how good are the recall and precision of a model for a specific target class: higher AUC values reflect higher TPR values and lower FPR values. Similar to 2, the AUC values correspond to the ROC curves for the attack class. To calculate the AUC in a continuous fashion, one needs to sort a given dataset by class probability and iterate through each observation. Because the sorted order of observations defines the resulting value of AUC, adding a new observation to the dataset forces the procedure to be repeated. As such, AUC cannot be directly computed on data streams, given the associated time and memory requirements. In [54], authors propose an efficient incremental algorithm that uses a sorted tree structure with a sliding window to compute the AUC using constant time and memory.

We use the MOA machine-learning library [55] to perform the analysis, including both the hyperparameter calibration (using a grid-search procedure) and the model training and evaluation. MOA is specifically designed for stream-based machine-learning approaches. Finally, to limit redundancy, we evaluate ADAM using ADWIN through prequential k-fold cross-validation, and ADAM using FIXWIN through AUC prequential evaluation. Nevertheless, drawn conclusions are the same as those obtained by running the full benchmark with all models and both evaluation strategies.



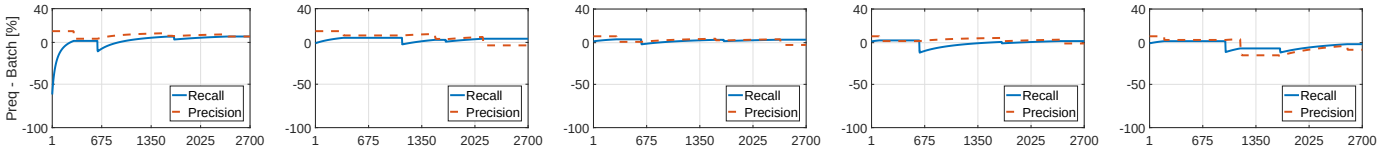
(a) *k-NN.* (b) *HAT.* (c) *ARF.* (d) *SGD.*

Figure 5: *Prequential 10-fold cross-validation accuracy evaluation.* Diagrams show prequential 10-fold CV results for each algorithm for each attack type. Concept drifts detected by the Page-Hinkley test are marked with dashed lines.

1) ***Concept-Drift Detection****:* We first study the variation of the statistical properties of the considered dataset, in particular detecting concept drifts with the Page-Hinkley test. Figure 4 depicts the cumulative number of changes observed in the dataset, as well as the times when those changes are detected. The test detects 14 abrupt changes during the total measurement time span. The frequency of changes significantly increases in the last third of the dataset, with more than 10 changes detected in the last four days. Concept drifts occur from modifications of the underlying characteristics of the prediction target. Concept drifts can be used to explain sudden shifts in the performance of algorithms as depicted in Figure 4.

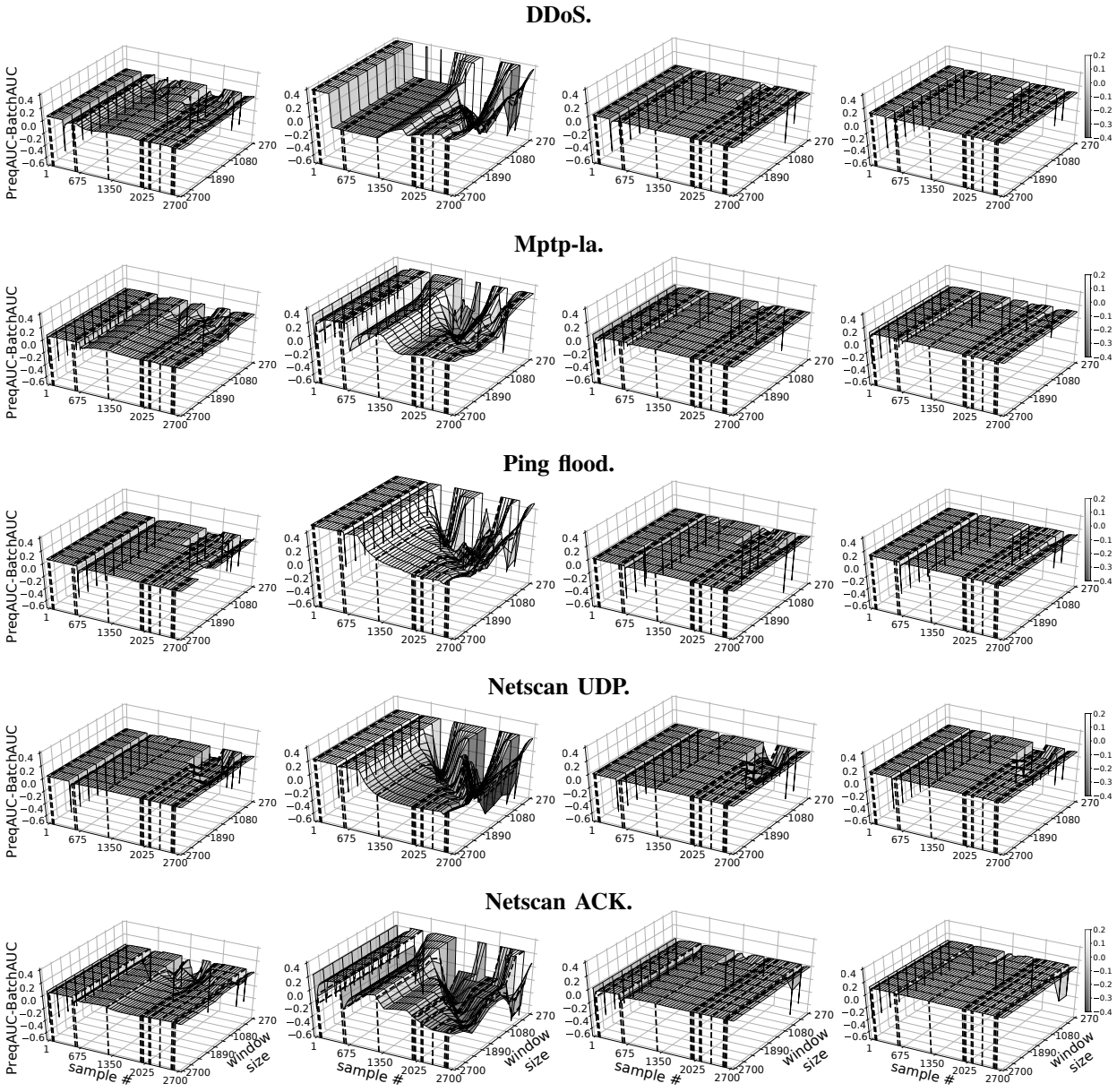
2) ***Performance Evaluation – ADWIN****:* We evaluate the performance of ADAM using ADWIN on top of the learning algorithms in five binary-classification scenarios, one for each attack type, resulting in five sets of results for each tested algorithm. Figure 5 reports the performance results for each attack type, considering detection accuracy as performance metric – i.e., recall for the attack class – and using the batch-algorithm accuracy as baseline. The prequential 10-fold-CV performance evaluation shows that both the ARF and SGD models rapidly converge to the batch-based accuracy results, with minimum performance variations under concept drifts, and with a slightly better performance for the SGD model, this one even outperforming the batch-based performance.

On the other hand, both the k-NN and HAT models do not show any apparent convergence and results tend to oscillate around the batch-algorithm baseline. In the case of HAT, we can appreciate the correlation between the detected concept drifts and the performance variations of the model. Interest-



(a) DDoS. (b) Mptp-la. (c) Ping flood. (d) Netscan UDP. (e) Netscan ACK.

Figure 6: *Prequential 10-fold cross-validation recall (detection accuracy) and precision evaluation.* Diagrams show prequential 10-fold CV results for the best algorithm (SGD) for each attack type.



(a) *k-NN.* (b) *HAT.* (c) *ARF.* (d) *SGD.*

Figure 7: *Prequential AUC evaluation.* Diagrams show the dependency of the prequential AUC results on the evaluation-window size, for each attack type. Concept drifts are marked with dashed lines.

ingly, the HAT model is the one achieving the highest accuracy of the four models – up to an improvement of 40 percentage points with respect to the baseline –, but cannot maintain such a performance constantly in time, with significant deterioration. This scenario shows the delicate challenge to handle trade-off between model performance, learning frequency, and forgetting of past measurements.

To complement the evaluation, Figure 6 re-evaluates the best performing model (i.e., SGD), on the five attack types, additionally evaluating the precision of the model for the attack

dataset	ε	η	initial θ	budget	ρ^+	ρ^-
MAWI	2.5%	0.01	0.9	0.05	1	−1
Woodcover	5%	0.02	0.9	0.01	1	−1

Table II: RAL hyperparameters, selected by grid search.

class. As before, values are normalized to the batch/offline performance (cf. Figure 2(e)). Recall results match those presented in Figure 5 for SGD, confirming a sustained high detection accuracy over time. In addition, the sustained high precision for all attack types also confirms the low false-alarm rates achieved by the streaming model, observed in Figure 2(e). As we show next in Figure 7, the area under the ROC curve (AUC), both SGD and ARF maintain a high detection accuracy and low false-alarm rate, with a consistent and sustained-over-time AUC matching the one achieved offline (cf. Figure 2).

*3) **Performance Evaluation - FIXWIN**:* We now take the FIXWIN strategy as basis for the analysis, using prequential-AUC evaluation. We evaluate different sliding-window sizes, by setting them to a fraction of the total dataset size: {10%,20%,...,100%}, i.e., {270,540,...,2700} samples, considering 10 independent runs. Figure 7 shows the prequential-AUC values along time (number of samples), with the batch-based AUC values as baseline (cf. Figure 2), for the 10 tested window sizes. Recall that a high AUC value shows that the underlying model achieves high detection accuracy and low false-alarm rates. For all four models, we first observe how increasing the window size smooths performance variations along time. However, as observed before in the case of ADWIN, using smaller or more fine-tuned window sizes can translate into better performance; for example, the HAT model achieves a performance gain of up to 40 percentage points (with respect to the baseline) when using 10%-dataset window size. The window size allows to track long or short-term changes better, depending on the tuning.

As before, we see how both ARF and SGD are highly robust to concept drifts and converge for almost all window sizes; this also happens, to a lesser extent, for the k-NN model, which finds convergence and robustness for window sizes above 40% of the dataset size. The performance of HAT also starts converging for longer window sizes, but with an important performance degradation for some attack types, directly implying that keeping past history under concept drifts might negatively impact results.

D. RAL Evaluation Strategy and Performance

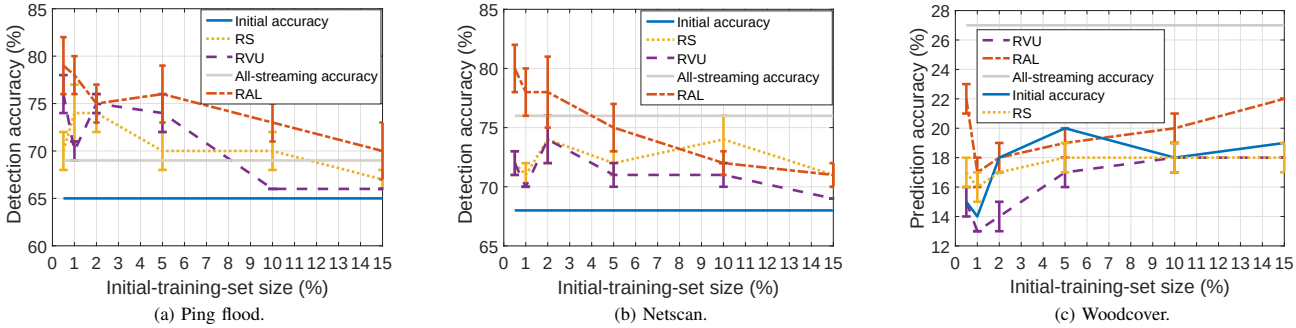
To showcase the performance of RAL, we evaluate and compare it to a state-of-the-art algorithm for stream-based active learning, as well as against a very basic random-sampling approach (RS). We compare RAL to the Randomized Variable Uncertainty (RVU) technique proposed in [28], [29], as this approach also heavily relies on the uncertainty of the underlying machine-learning models to take the querying decisions. Besides the MAWI datasets, we also use a subset of the widely used Forest Covertype data (https://archive.ics.uci.edu/ml/datasets/Covertype) to showcase RAL's generalization properties in a wider range of application domains. This dataset contains samples labeled with different forest cover types, represented by cartographic variables. Different from MAWI, where datasets have a binary label and the analysis focuses on the accuracy for the attack class (i.e., recall), the Woodcover dataset consists of seven different classes reflecting the different wood cover types; therefore, accuracy evaluations for this dataset do not correspond to a specific class, but rather to the overall accuracy across all seven classes. As the data is perfectly balanced among classes, results are comparable and not biased. We use the term *prediction accuracy* when referring to Woodcover results.

For each benchmarked algorithm, we proceed as follows: first, we subdivide the datasets into three consecutive, disjoint parts, i.e., the initial training set, the streaming set, and the validation set. The validation set consists of the last 30% of the dataset, the initial training set is a variable fraction of the first samples (varying between the first 0.5%, 1%, 2%, 5%, 10%, and 15%), and the streaming set includes the remaining samples. We train a model on the initial training set and check its detection accuracy on the validation set – we refer to this as the *initial accuracy*. Next, we run the specific active-learning algorithm on the streaming set and let it pick the samples it decides to learn from. To emulate a stream-learning scenario, we retrain the model each time a new label is queried. Finally, the reported accuracy for each algorithm corresponds to the application of a final model, trained on the initial training set plus the selected samples, on the validation set.

We implement the budget mechanism presented in [28] for both RAL and RVU, based on the ratio between the number of queries and the total number of samples observed so far; the system can issue queries to the oracle as long as this ratio is below a certain budget. For RS, we use a budget indicating the exact number of samples to ask feedback for. For each attack type, we set it to the highest average number of queried samples by either RAL or RVU among all the tests with all the considered initial-training-set sizes.

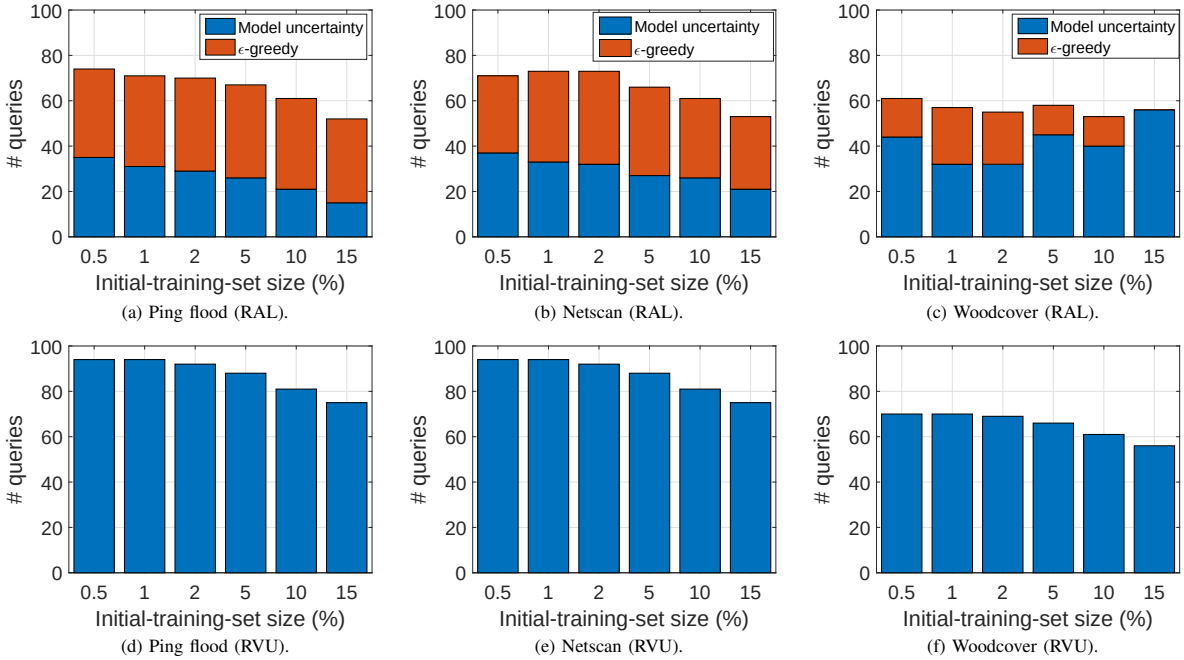
Similarly to ADAM's evaluation, all tests are repeated 10 times, and we report both average detection accuracy and standard errors. For RAL, we indicate the average number of queries performed due to the uncertainty of the underlying model, as well as those issued through the ε-greedy mechanism. For comparison purposes, we also report the average number of queries issued by RVU. The hyperparameter values of RAL are chosen by grid search on the corresponding training sets, within the ranges prescribed in Section IV. The used values are indicated in Table II. RAL's and RVU's budgets are set to the same value of 0.05 for all the experiments. In the case of RVU, we set its parameters based on those recommended in [29]. We limit the set of results in MAWI to only two attack types, namely ping flood and UDP netscan.

Figures 8 and 9 show the obtained results in terms of detection accuracy and number of queries, respectively, for (a) ping-flood, (b) UDP-netscan, and (c) Woodcover datasets. In Figure 8, the reported *all-streaming accuracy* (gray line) refers to the detection accuracy obtained by the model in case it queries all the samples seen in the stream. We apply the



(a) Ping flood. (b) Netscan. (c) Woodcover.

Figure 8: Detection accuracy for RAL, RVU, and RS. For each of the tested datasets, RAL outperforms both techniques.



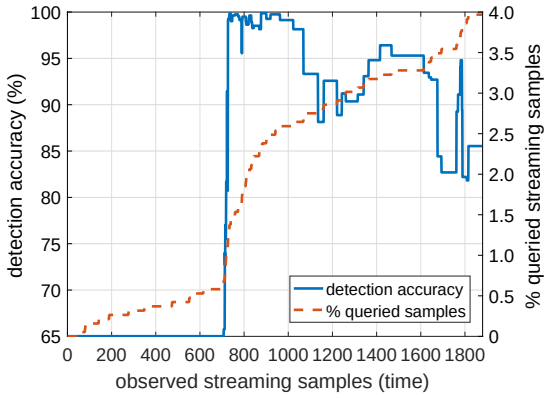
(a) Ping flood (RAL). (b) Netscan (RAL). (c) Woodcover (RAL).

(d) Ping flood (RVU). (e) Netscan (RVU). (f) Woodcover (RVU).

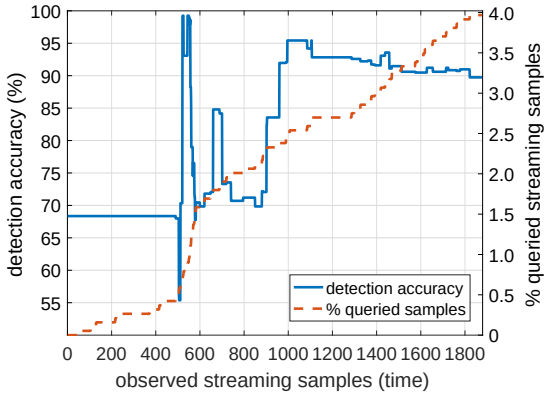
Figure 9: Number of queries issued by RAL (top) and RVU (bottom). RAL achieves better accuracy, querying fewer samples.

committee version of RAL on the detection of attacks, and the single classifier version of RAL in the wood-covertype set. The committee is a voting classifier composed of a k-NN model with $k = 5$, a decision tree, and a random forest with 10 trees, whereas the single classifier is a 10-tree random forest. We use the same models for RVU and RS in the corresponding datasets, i.e., committee for attacks and single classifier for Woodcover, relying on the same machine-learning algorithms as RAL. The accuracy plots show that RAL outperforms both RVU and RS on average, in all the datasets. A striking example is the result for the UDP-netscan detection, where RAL obtains accuracies which are almost 10 percentage points higher than the ones of RVU and RS for the two smallest initial-training-set sizes. In the case of Woodcover, RAL still yields better accuracies than both RVU and RS, even though this prediction task seems very challenging, as accuracy values are very low. To our surprise, RVU is often outperformed by RS. Finally, the ping-flood detection analysis shows that the three approaches often yield an accuracy higher than the *all-streaming* one, underlining that learning from the entire data stream does not necessarily translate into better performance. This could be explained by the high number of concept drifts in the data. The initial accuracy is constant for the two different MAWI attack subsets. This is due to the fact that the first 15% of these datasets consist of points with the same label (more precisely, they represent an attack).

When it comes to the number of queried samples, Figure 9 shows that RAL queries, on average, significantly less often than RVU, and especially for the detection of networks attacks (where more concept drifts occur) – between 20% and 25% fewer queries. A non-negligible part of these queries are due to the model's uncertainty, suggesting that the samples picked by RAL are wisely chosen. The results also highlight that the ε-greedy policy is very useful, as the additional exploration capability helps better deal with the concept drifts in the data, contributing to the better results showed in Figure 8. Finally, the number of samples/labels queried by RAL represents less than 4% of the total number of streaming samples, also showing how much one can save in terms of required labeling for training.



(a) Ping flood.



(b) UDP netscan.

Figure 10: RAL's detection accuracy temporal convergence.

We also study the convergence of RAL's attack-detection performance with respect to the evolution of the streaming samples (i.e., time), for the two MAWI attack datasets. More precisely, we evaluate RAL on the validation set after a new sample is queried. We set the initial training-set size to the first 0.5% of the data: according to Figure 8, such a small initial training-set provides the best results. Figure 10 reports the accuracy convergence for the ping-flood and netscan detection, along with the temporal evolution of the number of queried samples. We observe that the detection accuracy is not clearly converging in the two scenarios: the ping-flood-detection performance seems to converge to 90%, while there does not seem to be any convergence for the netscan case. This is not surprising, considering that these datasets present multiple concepts drifts and are very dynamic. We investigated the reasons behind the sharp accuracy increase in both evaluations, and found that they are highly correlated with queries issued by the committee – not the ε-greedy scenario. This analysis confirms that acquiring the labels for which the models have a low confidence in their prediction is indeed a good strategy for active learning, and in particular also for RAL. The degradation of the detection performance is likely due to the acquisition of noisy points in the dataset and to the concept drifts. The significant decrease in accuracy is mostly caused by samples queried by random exploration (ε-greedy) and not by RAL's committee, even though this mechanism also often provides performance boost by forcing RAL to explore the data space.

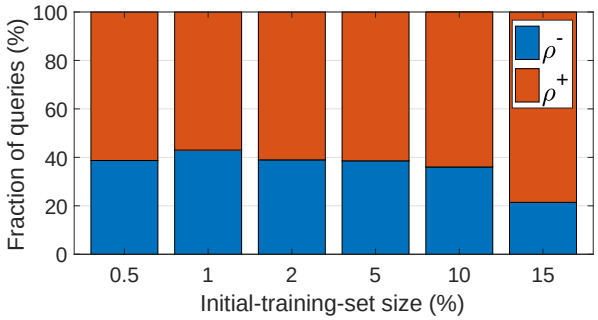


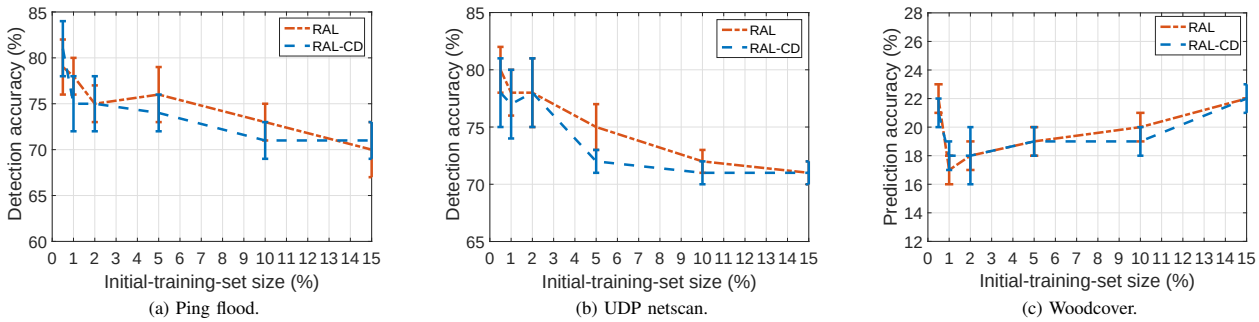
Figure 11: Proportion of obtained rewards vs. obtained penalties by RAL – wood-covertype dataset.

Finally, Figure 11 reports the fraction of positive and negative rewards obtained by the reinforcement-learning loop, in this case for the woodcover set. Results show that, for each different initial-training-set size, the fraction of useful queries is above 60%, further confirming the expected theoretical results regarding the total reward, cf. Section IV-B.

Based on these results, one could wonder whether the performance gain by RAL is worth the complexity of the system. Even though the accuracy gain might not be very significant, RAL's querying strategy has additional advantages over the two other techniques. For instance, RS does not consider the uncertainty of the model nor the usefulness of the queries, meaning that there is a risk to miss interesting samples. Indeed, querying the ground truth when the model is uncertain helps discover under-explored regions where to learn from, and RAL is additionally guided by its reward mechanism. In the specific case of the MAWI dataset, RS would probably miss interesting attack samples, while RAL has a higher chance of querying the ground truth for these data samples and better learn how to detect attacks. When it comes to RVU, another advantage of RAL over that algorithm, besides its better performance shown above, is that the querying decisions are also influenced by the informativeness of all past queries, not only by their sheer execution; RVU does not take that information into account at all, and thus it risks querying unnecessary samples too often. This is especially problematic if querying is very expensive, or if the oracle has only limited budget/availability.

E. Integrating ADAM & RAL

To conclude the study, we evaluate whether extending RAL with explicit concept-drift detection, as ADAM, could provide further improvements in terms of performance and adaptability. Inspired by ADAM and based on previous work [21], [22], we use ADWIN to detect shifts in the reward signal of RAL, which is a good proxy for detection of concept drifts in the input data. The reasoning behind this is simple: in case RAL's querying behavior starts to deteriorate by accumulating penalties, i.e., a series of ρ^-, we need to retrain the underlying models with only the most recent data to eliminate the outdated data points from the training set; conversely, if RAL starts to perform well by accumulating a series of ρ^+, we might



(a) Ping flood. (b) UDP netscan. (c) Woodcover.

Figure 12: Detection-accuracy comparison between RAL and RAL with explicit Concept-Drift detection (RAL-CD).

improve and strengthen this behavior by removing older data. We rely on the scikit-multiflow library [56] to implement ADWIN into RAL. We refer to this approach as RAL-CD – RAL with explicit Concept-Drift detection.

We evaluate RAL-CD by proceeding in the same way as for the evaluation of RAL. In particular, we compare RAL-CD against RAL on the same three datasets. Figure 12 reports the obtained results. The number of concept drifts detected by ADWIN in RAL is generally very low: in fact, in most of the repetitions, ADWIN detected only one change in the reward pattern for the whole MAWI streams, and almost no changes when it comes to the Woodcover dataset. RAL-CD's detection accuracy is slightly worse than that of RAL. Intuitively, this shows that the former version of RAL, i.e., without any explicit drift detection, already selects the best samples that yield high accuracy. Results reveal that using smaller training sets with RAL-CD yields worse performance than with RAL. As an overall conclusion, the outcomes of this evaluation show that the drift detector does not improve RAL's predictive performance, underlining that the samples selected by RAL are already wisely chosen. However, we hypothesize that RAL-CD might actually prove useful in the case of longer and even more dynamic streams, with more pronounced drifts: learning an accurate model on both pre- and post-drift data might not be feasible, and removing the pre-drift data could bring a distinctive advantage to RAL. We are currently considering a more comprehensive evaluation of RAL-CD on other datasets.

VI. Concluding Remarks

Dynamic and adaptive-memory-based learning seems to be a promising learning strategy to adapt to very dynamic environments, where concept drifts occur often. This is a common scenario when dealing with online network-traffic-monitoring applications. We have introduced and evaluated ADAM and RAL, two stream-based machine-learning approaches to tackle important challenges when dealing with data streams. We have shown that ADAM allows to track transient changes and concept drifts along time. Indeed, using ADAM, adaptive learning algorithms can continuously achieve high detection accuracy over dynamic network data streams, when dynamically adapting their learning pace and memory to changes in the underlying statistics of the samples. We have confirmed that both adaptive random forests and SVM through stochastic gradient descent are better suited for the studied problem, especially in terms of robustness to concept drifts and convergence of results. We have also introduced RAL, a novel Reinforced stream-based Active-Learning approach to tackle the challenges of stream-based active learning, i.e., selecting the most valuable sequentially incoming samples to reduce the amount of learning data to label, using reinforcement-learning principles. RAL does not only learn from the data stream, but also from the relevance of its own querying decisions. RAL provides a completely different exploration-exploitation trade-off than existing algorithms. Evaluations have shown that RAL provides very promising results, outperforming state-of-the-art techniques, providing higher accuracies with less ground truth. As an additional contribution, we make RAL freely available on GitHub.

References

[1] G. Li et al., "Detecting cyberattacks in industrial control systems using online learning algorithms," in *Neurocomputing*, vol. 364, pp. 338–348, 2019.

[2] B. Settles, "Active learning literature survey," University of Wisconsin-Madison, Tech. Rep., 2010.

[3] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab : Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking," in *6th ACM CoNEXT Conference*, 2010.

[4] S. Wassermann, T. Cuvelier, P. Mulinka, and P. Casas, "ADAM & RAL: Adaptive Memory Learning and Reinforcement Active Learning for Network Monitoring," in *15th International Conference on Network and Service Management (CNSM)*, 2019.

[5] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. E. Solano, and O. M. C. Rendon, "A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 16:1–16:99, 2018.

[6] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 15:1–15:58, jul 2009.

[7] M. Ahmed, A. Naser Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, no. C, pp. 19–31, 2016.

[8] W. Zhang, Q. Yang, and Y. Geng, "A survey of anomaly detection methods in networks," in *Computer Network and Multimedia Technology CNMT*, 2009.

[9] J. Vanerio and P. Casas, "Ensemble-learning approaches for network security and anomaly detection," in *ACM SIGCOMM Big-DAMA Workshop*, 2017.

[10] P. Casas, J. Vanerio, and K. Fukuda, "Gml learning, a generic machine learning model for network measurements analysis," in *13th International Conference on Network and Service Management (CNSM)*, 2017.

[11] V. Carela-Español, P. Barlet-Ros, A. Bifet, and K. Fukuda, "A streaming flow-based technique for traffic classification applied to 12+ 1 years of internet traffic," *Telecommunication Systems*, vol. 63, no. 2, pp. 191–204, 2016.

[12] P. Domingos and G. Hulten, "Mining High-Speed Data Streams," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.

[13] P. M. Domingos and G. Hulten, "Catching up with the data: Research issues in mining data streams." in *Workshop on Research Issues in Data Mining and Knowledge Discovery DMKD*, 2001.

[14] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42–47, 2005.

[15] G. Hulten, P. Domingos, and L. Spencer, *Mining massive data streams*. University of Washington, 2005.

[16] J. Gama, R. Sebastião, and P. P. Rodrigues, "Issues in evaluation of stream learning algorithms," in *15th ACM SIGKDD Conference*, 2009.

[17] ——, "On evaluating stream learning algorithms," *Machine learning*, vol. 90, no. 3, pp. 317–346, 2013.

[18] D. Bouneffouf, "Online learning with Corrupted context: Corrupted Contextual Bandits," *arXiv preprint arXiv:2006.15194*, 2020.

[19] T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning from streaming data with concept drift and imbalance: an overview," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 89–101, 2012.

[20] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM Conference*, 2007.

[21] A. Bifet, G. Holmes, B. Pfahringer, and R. Gavaldà, "Improving adaptive bagging methods for evolving data streams," in *Advances in Machine Learning, First Asian Conference on Machine Learning ACML*, 2009.

[22] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis IDA*, 2009.

[23] A. Bifet, B. Pfahringer, J. Read, and G. Holmes, "Efficient data stream classification via probabilistic adaptive windows," in *28th Annual ACM Symposium on Applied Computing, SAC*, 2013.

[24] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence – SBIA*, 2004.

[25] M. Baena-Garc, J. del Campo Ávila, A. Bifet, R. Gavald, and R. Morales-Bueno, "Early Drift Detection Method," in *Fourth International Workshop on Knowledge Discovery from Data Streams*, 2006.

[26] V. Losing, B. Hammer, and H. Wersing, "Knn classifier with self adjusting memory for heterogeneous concept drift," in *IEEE 16th International Conference on Data Mining (ICDM)*, 2016.

[27] G. Widmer and M. Kubat, "Learning in the Presence of Concept Drift and Hidden Contexts," in *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996.

[28] I. Žliobaitė, A. Bifet, B. Pfahringer, and G. Holmes, "Active learning with evolving streaming data," in *Machine Learning and Knowledge Discovery in Databases*. Springer Berlin Heidelberg, 2011, pp. 597–612.

[29] ——, "Active learning with drifting streaming data," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 27–39, January 2014.

[30] W. Xu, F. Zhao, and Z. Lu, "Active learning over evolving data streams using paired ensemble framework," in *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*, February 2016, pp. 180–185.

[31] D. Ienco, A. Bifet, I. Žliobaitė, and B. Pfahringer, "Clustering based active learning for evolving data streams," in *Discovery Science*, 2013, pp. 79–93.

[32] B. Krawczyk, "Active and adaptive ensemble learning for online activity recognition from data streams," in *Knowledge-Based Systems*, vol. 138, pp. 69–78, 2017.

[33] S. Sinha, S. Ebrahimi, and T. Darrell. "Variational adversarial active learning," in *IEEE International Conference on Computer Vision*, 2019.

[34] Y. Baram, R. El-Yaniv, and K. Luz, "Online choice of active learning algorithms," *J. Mach. Learn. Res.*, vol. 5, pp. 255–291, dec 2004.

[35] W.-N. Hsu and H.-T. Lin, "Active learning by learning," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, pp. 2659–2665.

[36] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2002.

[37] L. Song, "Stream-based online active learning in a contextual multi-armed bandit framework," *arXiv preprint arXiv:1607.03182*, 2016.

[38] L. Song and J. Xu, "A contextual bandit approach for stream-based active learning," *arXiv preprint arXiv:1701.06725*, 2017.

[39] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap, "One-shot learning with memory-augmented neural networks," *CoRR*, vol. abs/1605.06065, 2016.

[40] P. Bachman, A. Sordoni, and A. Trischler, "Learning algorithms for active learning," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 301–310.

[41] M. Fang, Y. Li, and T. Cohn, "Learning how to active learn: A deep reinforcement learning approach," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, 2017, pp. 595–605.

[42] K. Pang, M. Dong, Y. Wu, and T. M. Hospedales, "Meta-learning transferable active learning policies by deep reinforcement learning," *CoRR*, vol. abs/1806.04798, 2018.

[43] K. Konyushkova, R. Sznitman, and P. Fua, "Learning active learning from real and synthetic data," *CoRR*, vol. abs/1703.03365, 2017.

[44] M. Woodward and C. Finn, "Active one-shot learning," *CoRR*, vol. abs/1702.06559, 2017.

[45] H. Huang et al., "On the improvement of reinforcement active learning with the involvement of cross entropy to address one-shot learning problem," *PloS one* vol. 14, no. 6, 2019.

[46] A. Puzanov, S. Zhang, and K. Cohen, "Deep reinforcement one-shot learning for artificially intelligent classification in expert aided systems," *Engineering Applications of Artificial Intelligence*, vol. 91, 2020.

[47] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, vol. 106, no. 9-10, pp. 1469–1495, 2017.

[48] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM CSUR*, vol. 46, no. 4, p. 44, 2014.

[49] E. S. PAGE, "Continuous Inspection Schemes," *Biometrika*, vol. 41, no. 1-2, pp. 100–115, 06 1954.

[50] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[51] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2000.

[52] Y. Guermeur, "Vc theory of large margin multi-category classifiers," *Journal of Machine Learning Research*, vol. 8, no. November, pp. 2551–2594, 2007.

[53] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *Proceedings of the 21th ACM SIGKDD Conference*. ACM, 2015, pp. 59–68.

[54] D. Brzezinski and J. Stefanowski, "Prequential auc: properties of the area under the roc curve for data streams with concept drift," *Knowledge and Information Systems*, vol. 52, no. 2, pp. 531–562, 2017.

[55] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *Journal of Machine Learning Research*, vol. 11, no. May, pp. 1601–1604, 2010.

[56] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *Journal of Machine Learning Research*, vol. 19, pp. 72:1–72:5, 2018.

BIOGRAPHIES

Sarah Wassermann is currently a third-year PhD student at TU Wien, doing research in network measurements, in particular in the field of QoE, and machine learning. She earned her Master's degree in 2017 from the university of Liège (ULiège) in Belgium, where she also obtained her Bachelor's degree in 2015. Her goal is to conceive intelligent systems which make the Internet smarter and able to face demanding users and an ever-growing volume of heterogeneous network traffic.

Thibaut Cuvelier is a third-year PhD student at CentraleSupélec (France), doing research at the intersection of mathematical optimization and machine learning. He earned his Master's degree in 2015 from the university of Liège (Belgium). His current research interests include bandit optimization, especially combinatorial bandits, and applications of optimization in computer networks, more precisely in routing and related cut problems.

Pedro Casas is Senior Scientist in AI/ML for Networking at the AIT Austrian Institute of Technology in Vienna. He received an Electrical Engineering degree from Universidad de la República, Uruguay in 2005, and a Ph.D. degree in Computer Science from Télécom Bretagne in 2010. He was Postdoctoral Research at the LAAS-CNRS in Toulouse from 2010 to 2011, and Senior Researcher at the Telecommunications Research Center Vienna (FTW) from 2011 to 2015. His work focuses on machine-learning-based approaches for Networking, big data analytics and platforms, Internet network measurements, network security and anomaly detection, as well as Internet QoE monitoring. He has published more than 180 Networking research papers in major international conferences and journals, received 14 awards for his work - including 7 best paper awards. He is general chair for different actions in network measurement and analysis, including the IEEE ComSoc ITC Special Interest Group on Network Measurements and Analytics.

Pavol Mulinka is a PhD candidate at the CTU Czech Technical University in Prague. He is a machine learning enthusiast, data scientist, freelance Python programmer and former network engineer. He is currently working as a researcher at the CTTC Research Center in Barcelona, and as a data science volunteer in Wikimedia Scoring team. He enjoys riding motorcycle, rock climbing, jogging, and yoga.

VII. Appendix – RAL Bounds Proof

In this section, we provide a formal proof for the bounds on the expected total reward as presented in Section IV-B. The specific bounds are as follows:

$$\begin{aligned}\mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\} &\leq T\left(\rho^+ + \rho^-\right) + T\left(\rho^+ - \rho^-\right)\beta' \\ \mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\} &\geq T\left(\rho^+ - \rho^-\right)\beta\end{aligned}$$

where β and β' are parameters defined later in this appendix; they depend on the prediction performance of the machine-learning models.

The proof is done in three consecutive steps, going from the simple version of a single binary classifier (Section VII-A), to a single multiclass classifier (Section VII-B), to a multiclass voting classifier (Section VII-C), the latter corresponding to the committee version of RAL. These three cases all have the same generic bound, with slightly different definitions of β and β'.

A. Expected Reward Analysis – Single Classifier

Let us analyze the expected total reward obtained by using RAL, i.e., $\mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\}$, where T denotes the number of samples in the considered data stream and r_n indicates the reward obtained for the n-th sample. In the following developments, we use these notations:

- $\hat{y}_n$ – n-th predicted value
- $\hat{p}_n$ – certainty of the model for the n-th prediction
- $\rho^{\pm}$ – reward and penalty obtained by RAL respectively; the reward ρ^+ must be nonnegative and the penalty ρ^- nonpositive
- $\mathcal{VC}$ – Vapnik-Chervonenkis dimension of the learner [51]
- θ_n – uncertainty threshold before having observed the n-th sample
- err_n – error rate of our classifier before having observed the n-th sample
- $\overline{err}_n$ – training error of model before having observed the n-th sample

The expected total reward writes $\mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\} = \sum_{n=1}^{T}\mathbb{E}\{r_n\}$. Based on the classical result of [51], we have the following bound:

$$f_n(\alpha) = \overline{err}_n + \sqrt{\frac{1}{N_n}\left[\mathcal{VC}\left(\log\frac{2N_n}{\mathcal{VC}} + 1\right) - \log\frac{\alpha}{4}\right]}$$

$$\mathbb{P}\left(err_n \leq f_n(\alpha)\right) = 1 - \alpha$$

where N_n denotes the training-set size for the underlying classifier at the n-th round (before observing the n-th sample) and α is a confidence level whose value lies in the interval $[0, 1]$. We can therefore write this probabilistic bound as:

$$\mathbb{P}\Big[\mathbb{P}(\hat{y}_n \neq y_n) \leq f_n(\alpha)\Big] = 1 - \alpha$$

This means that the probability of making a mistake can be written as:

$$\begin{aligned}\mathbb{P}[\hat{y}_n \neq y_n] &= \mathbb{P}[\hat{y}_n \neq y_n | \mathbb{P}(\hat{y}_n \neq y_n) \leq f_n(\alpha)] \\ &\times \mathbb{P}[\mathbb{P}(\hat{y}_n \neq y_n) \leq f_n(\alpha)] \\ &+ \mathbb{P}[\hat{y}_n \neq y_n | \mathbb{P}(\hat{y}_n \neq y_n) > f_n(\alpha)] \\ &\times \mathbb{P}[\mathbb{P}(\hat{y}_n \neq y_n) > f_n(\alpha)] \\ &= \mathbb{P}[\hat{y}_n \neq y_n | \mathbb{P}(\hat{y}_n \neq y_n) \leq f_n(\alpha)]\ (1-\alpha) \\ &+ \mathbb{P}[\hat{y}_n \neq y_n | \mathbb{P}(\hat{y}_n \neq y_n) > f_n(\alpha)]\ \alpha\end{aligned}$$

Its upper and lower bounds are thus:

$$0 + \alpha\ f_n(\alpha) \leq \mathbb{P}[\hat{y}_n \neq y_n] \leq (1-\alpha)\ f_n(\alpha) + \alpha \times 1$$

For the next proofs, we will require bounds on the probability of the certainty of the model being less than a threshold. Unfortunately, to the best of our knowledge, no generic result exists for the probability distribution of these certainties, which leads to very loose bounds:

$$0 \leq \mathbb{P}[\hat{p}_n \leq \theta_n] \leq 1$$

For the following steps, we rely on classical results in probability theory, namely the union bound and Fréchet's inequality. For two probabilistic events A and B, be they independent or not, the following bounds hold:

$$\mathbb{P}(A \wedge B) \leq \mathbb{P}(A) + \mathbb{P}(B)$$

$$\mathbb{P}(A \wedge B) \geq \max\{0, \quad \mathbb{P}(A) + \mathbb{P}(B) - 1\}$$

We have that $\mathbb{E}\{r_n\} = \sum_{r\in\mathcal{R}} r \times \mathbb{P}(r_n = r)$ with $\mathcal{R} = \{\rho^+, \rho^-\}$ being the set of all possible reward values. As RAL does not obtain any reward in the ε-scenario, it can be ignored. Therefore, we have the following decomposition of the expectation and a generic upper bound:

$$\begin{aligned}\mathbb{E}\{r_n\} &= \underbrace{\rho^+}_{\geq 0}\ \underbrace{\mathbb{P}[\hat{p}_n \leq \theta_n \wedge \hat{y}_n \neq y_n]}_{\leq(\mathbb{P}[\hat{p}_n \leq \theta_n] + \mathbb{P}[\hat{y}_n \neq y_n])} \\ &+ \underbrace{\rho^-}_{\leq 0}\ \underbrace{\mathbb{P}[\hat{p}_n \leq \theta_n \wedge \hat{y}_n = y_n]}_{\geq(\mathbb{P}[\hat{p}_n \leq \theta_n] + \mathbb{P}[\hat{y}_n = y_n] - 1)} \\ &\leq \mathbb{P}[\hat{p}_n \leq \theta_n]\left(\rho^+ + \rho^-\right) + \mathbb{P}[\hat{y}_n \neq y_n]\ \rho^+ \\ &+ [1 - \mathbb{P}[\hat{y}_n \neq y_n]]\ \rho^- - \rho^- \\ &\leq \mathbb{P}[\hat{p}_n \leq \theta_n]\left(\rho^+ + \rho^-\right) + \mathbb{P}[\hat{y}_n \neq y_n]\left(\rho^+ - \rho^-\right)\end{aligned}$$

Finally, the upper bound on the expected total reward, under the assumption that both $(\rho^+ + \rho^-)$ and $(\rho^+ - \rho^-)$ are nonnegative, is:

$$\mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\} \leq T\left(\rho^+ + \rho^-\right) + T\left(\rho^+ - \rho^-\right)\underbrace{\left[(1-\alpha)\ f_n(\alpha) + \alpha\right]}_{\beta'}$$

If these two assumptions do not hold, a similar bound can still be achieved:

First, suppose that $\rho^+ + \rho^- \geq 0$ and $\rho^+ - \rho^- \leq 0$. In this case, the only solution is to have $\rho^+ = \rho^- = 0$, thus trivially $\mathbb{E}\{\sum_{n=1}^{T} r_n\} = 0$.

Second, suppose that, conversely, $\rho^+ + \rho^- \leq 0$ and $\rho^+ - \rho^- \geq 0$. These assumptions lead to:

$$\mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\} \leq T\left(\rho^+ - \rho^-\right) \underbrace{\left[(1-\alpha)\ f_n(\alpha) + \alpha\right]}_{\beta'}$$

Third, the case where both $\rho^+ + \rho^- \leq 0$ and $\rho^+ - \rho^- \leq 0$ should not be studied further, because that would imply that $\rho^+ \leq 0$, which violates the defined range of allowed values for ρ^+ (in case $\rho^+ = 0$, we must have $\rho^- = 0$).

As a next step, we derive a lower bound of the expected total reward, with a very similar reasoning. First, the expected reward can be decomposed as:

$$\begin{aligned}
\mathbb{E}\{r_n\} &= \underbrace{\rho^+}_{\geq 0}\ \underbrace{\mathbb{P}\left[\hat{p}_n \leq \theta_n \wedge \hat{y}_n \neq y_n\right]}_{\geq (\mathbb{P}[\hat{p}_n \leq \theta_n] + \mathbb{P}[\hat{y}_n \neq y_n] - 1)} \\
&+ \underbrace{\rho^-}_{\leq 0}\ \underbrace{\mathbb{P}\left[\hat{p}_n \leq \theta_n \wedge \hat{y}_n = y_n\right]}_{\leq (\mathbb{P}[\hat{p}_n \leq \theta_n] + \mathbb{P}[\hat{y}_n = y_n])} \\
&\geq \mathbb{P}\left[\hat{p}_n \leq \theta_n\right]\left(\rho^+ + \rho^-\right) \\
&+ \left(\mathbb{P}\left[\hat{y}_n \neq y_n\right] - 1\right)\left(\rho^+ - \rho^-\right)
\end{aligned}$$

Eventually, if $\rho^+ \pm \rho^- \geq 0$, the expected total reward is at least $T\left(\rho^+ - \rho^-\right)\left[\alpha\ f_n(\alpha) - 1\right]$, i.e., $\beta = \alpha\ f_n(\alpha) - 1$. Conversely, if $\rho^+ + \rho^- \leq 0$ and $\rho^+ - \rho^- \geq 0$, the lower bound is $T\left(\rho^+ - \rho^-\right)\left[\alpha\ f_n(\alpha) - 2\right]$, i.e., $\beta = \alpha\ f_n(\alpha) - 2$.

B. Generalization to the Multiclass Case

The VC dimension makes no more sense when the classification problem includes multiple classes. There have been several generalizations thereof, for instance the covering number $\mathcal{N}^{(p)}\left(\gamma/4, \Delta_\gamma \mathcal{G}, 2\,N_n\right)$ [52], where $\Delta_\gamma \mathcal{G}$ is the set of classification margins obtained by any classifier of the family $\mathcal{G}$ in the known N_n data points (if a margin is larger than γ, it is clipped to γ). $\overline{err}_{\gamma,n}$ is the number of misclassifications, where an element is misclassified if its margin is less than γ. With a margin $\gamma \in \mathbb{R}_0^+$, a real number $\Gamma \in \mathbb{R}_0^+$ ($\gamma \leq \Gamma$), and the previously defined notations, the following bound on the generalization error holds:

$$\begin{aligned}
f_n(\alpha, \gamma) &= \overline{err}_{\gamma,n} + \frac{1}{N_n} \\
&+ \sqrt{\frac{2}{N_n}\left[\log\left(2\,\mathcal{N}^{(p)}\left(\frac{\gamma}{4}, \Delta_\gamma \mathcal{G}, 2\,N_n\right)\right) - \log\frac{2\,\Gamma}{\alpha\,\gamma}\right]}
\end{aligned}$$

$$\mathbb{P}\left(err_n \leq f_n(\alpha, \gamma)\right) = 1 - \alpha$$

Notation is taken directly as defined in [52].

Considering that $\rho^+ + \rho^- \geq 0$, the upper bound of the expected total reward can be computed as in the binary-classification problem:

$$\mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\} \leq T\left(\rho^+ + \rho^-\right) + T\left(\rho^+ - \rho^-\right) \underbrace{\left[(1-\alpha)\ f_n(\alpha, \gamma) + \alpha\right]}_{\beta'}$$

Similarly, the lower bound for a multiclass problem can be expressed as:

$$\mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\} \geq T\left(\rho^+ - \rho^-\right) \underbrace{\left[\alpha\ f_n(\alpha, \gamma) - 1\right]}_{\beta}$$

C. Committee Version

The mathematical developments for the committee version are very similar to the single classifier ones. First of all, the committee is still a classifier, and thus the same kind of bound applies on the probability of misclassifying. The only difference is that we have to take the VC dimension of the stacked classifiers instead of the one of the single classifier.

RAL asks the oracle for a label (and obtains the corresponding reward) if the weighted average of the decisions encourages it to query. We denote by $d_{i,n}$ the random variable indicating whether the i-th classifier decides to query the oracle or not, i.e., whether its certainty $\hat{p}_{i,n}$ for the n-th prediction is below the threshold θ_n (in case of querying, $d_{i,n} = 1$; otherwise, $d_{i,n} = 0$). $\alpha_{i,n}$ is the weight of the i-th classifier for the n-th sample; we have previously imposed that the sum of the weights must be one ($\sum_{i=1}^{C} \alpha_{i,n} = 1$ for each sample n). Thus, RAL asks when:

$$\sum_{i=1}^{C} \alpha_{i,n}\, d_{i,n} \geq \frac{1}{2}$$

For the upper bound, the previous developments still hold:

$$E\{r_n\} \leq \mathbb{P}\left[\sum_{i=1}^{C} \alpha_{i,n}\, d_{i,n} \geq \frac{1}{2}\right]\left(\rho^+ + \rho^-\right) + \mathbb{P}\left[\hat{y}_n \neq y_n\right]\left(\rho^+ - \rho^-\right)$$

Again, to the best of our knowledge, no generic result exists for a probability distribution of the querying decisions; we therefore have to resort to a very broad bound:

$$0 \leq \mathbb{P}\left[\sum_{i=1}^{C} \alpha_{i,n}\, d_{i,n} \geq \frac{1}{2}\right] \leq 1$$

Finally, the expected total reward is, if $\rho^+ \pm \rho^- \geq 0$, at most:

$$\mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\} \leq T\left(\rho^+ + \rho^-\right) + T\left(\rho^+ - \rho^-\right) \underbrace{\left[(1-\alpha)\ f_n(\alpha, \gamma) + \alpha\right]}_{\beta'}$$

Similarly, concerning the lower bound, we obtain, for the same reasons, the same lower bound as in the single classifier case. Specifically, if $\rho^+ \pm \rho^- \geq 0$,

$$\mathbb{E}\left\{\sum_{n=1}^{T} r_n\right\} \geq T\left(\rho^+ - \rho^-\right) \underbrace{\left[\alpha f_n(\alpha, \gamma) - 1\right]}_{\beta}$$