Pattern Recognition 45 (2012) 208–230

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Adaptive ROC-based ensembles of HMMs applied to anomaly detection

Wael Khreich [a,*], Eric Granger [a], Ali Miri [b], Robert Sabourin [a]

[a] Laboratoire d'imagerie, de vision et d'intelligence artificielle (LIVIA), École de technologie supérieure, Université du Québec, 1100 Notre-Dame Ouest, Montreal, QC, Canada
[b] School of Computer Science, Ryerson University, Toronto, Canada

ARTICLE INFO

Article history:
Received 18 November 2010
Received in revised form
4 June 2011
Accepted 14 June 2011
Available online 19 July 2011

Keywords:
Classification
Multi-classifier systems
Incremental learning
Adaptive systems
ROC
Information fusion
Hidden Markov models
Anomaly detection
Computer and network security

ABSTRACT

Hidden Markov models (HMMs) have been successfully applied in many intrusion detection applications, including anomaly detection from sequences of operating system calls. In practice, anomaly detection systems (ADSs) based on HMMs typically generate false alarms because they are designed using limited amount of representative training data. Since new data may become available over time, an important feature of an ADS is the ability to accommodate newly acquired data incrementally, after it has originally been trained and deployed for operations. In this paper, a system based on the receiver operating characteristic (ROC) is proposed to efficiently adapt ensembles of HMMs (EoHMMs) in response to new data, according to a learn-and-combine approach. When a new block of training data becomes available, a pool of base HMMs is generated from the data using a different number of HMM states and random initializations. The responses from the newly trained HMMs are then combined to those of the previously trained HMMs in ROC space using a novel incremental Boolean combination (incrBC) technique. Finally, specialized algorithms for model management allow to select a diversified EoHMM from the pool, and adapt Boolean fusion functions and thresholds for improved performance, while it prunes redundant base HMMs. The proposed system is capable of changing the desired operating point during operations, and this point can be adjusted to changes in prior probabilities and costs of errors. Computer simulations conducted on synthetic and real-world host-based intrusion detection data indicate that the proposed system can achieve a significantly higher level of performance than when parameters of a single best HMM are estimated, at each learning stage, using reference batch and incremental learning techniques. It also outperforms the learn-and-combine approaches using static fusion functions (e.g., majority voting). Over time, the proposed ensemble selection algorithms form compact EoHMMs, while maintaining or improving system accuracy. Pruning allows to limit the pool size from increasing indefinitely, thereby reducing the storage space for accommodating HMMs parameters without negatively affecting the overall EoHMM performance. Although applied for HMM-based ADSs, the proposed approach is general and can be employed for a wide range of classifiers and detection applications.

1. Introduction

Anomaly detection systems (ADSs) detect intrusions by monitoring for significant deviations from normal system behavior. Traditional host-based ADSs monitor for significant deviation in normal operating system calls—the gateway between user and kernel mode. ADSs designed with discrete hidden Markov models (HMMs) have been shown to produce a high level of accuracy [1,2]. A well-trained HMM provides a compact detector that captures the underlying structure of a process based on the temporal order of system calls, and detects deviations from normal system call sequences with high accuracy and tolerance to noise.

An ADS allows to detect novel attacks, however will typically generate false alarms due in large part to the limited amount of representative data [3]. Because the collection and analysis of training data to design and validate an ADS is costly, the anomaly detector will have an incomplete view of the normal process behavior, and hence misclassify rare normal events as anomalous. Substantial changes to the monitored environment, such as application upgrade or changes in users behavior, reduce the reliability of the detector as the internal model of normal behavior diverges with respect to the underlying data distribution. Since new training data may become

* Corresponding author. Tel.: +1 514 223 4404; fax: +1 514 396 8595.
E-mail addresses: khreichwael@yahoo.ca, wael.khreich@livia.etsmtl.ca (W. Khreich), eric.granger@etsmtl.ca (E. Granger), ali.miri@ryerson.ca (A. Miri), robert.sabourin@etsmtl.ca (R. Sabourin).

0031-3203/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2011.06.014

available over time, an ADS should accommodate newly acquired data, after it has originally been trained and deployed for operations, in order to maintain or improve system performance.

The incremental re-estimation of HMM parameters is a common approach to accommodate new data, although it raises several challenges. Parameters should be updated from new data without requiring access to the previous training data, and without corrupting previously learned models of normal behavior [4,5]. Standard techniques for estimating HMM parameters involve iterative batch learning algorithms [6,7], and hence require observing the entire training data prior to updating HMM parameters. Given new training data, these techniques can be costly since they involve restarting the HMM training using all cumulative training data. Alternatively, several efficient on-line learning techniques have been proposed to re-estimate HMM parameters continuously upon observing each new observation symbol or new observation sub-sequence from an infinite data stream [8,9]. In practice however, on-line learning using blocks comprising limited amount of data yields a low level of performance as one pass over each block is not sufficient to capture the phenomena [10]. It is also possible to adapt on-line learning techniques for incremental learning over several passes of each block, although it requires strategies to manage the learning rate [10]. Moreover, using a single HMM with a fixed number of states for incremental learning may not capture a representative approximation of the underlying data distribution due to the many local maxima in the solution space.

Ensemble methods have been recently employed to overcome the limitations faced with a single classifier system [11–14]. Theoretical and empirical evidence have shown that combining the outputs of several accurate and diverse classifiers is an effective technique for improving the overall system accuracy [11–17]. In general, designing an ensemble of classifiers involves generating a diverse pool of base classifiers [18–20], selecting an accurate and diversified subset of classifiers [21], and then combining their output predictions [12,14]. Most existing ensemble techniques aim at maximizing the overall ensemble accuracy, which assumes fixed prior probabilities and fixed misclassification costs. In many real-world applications, like anomaly detection, prior class distributions are highly imbalanced and misclassification costs may vary over time. Nevertheless, common techniques used for fusion, such as voting, sum or averaging, assume independent classifiers and do not consider class priors [16,22]. This paper focuses on HMM-based systems applied to anomaly detection, and in particular on the efficient adaptation of ensembles of HMMs (EoHMMs) over time, in response to new data.

The receiver operating characteristic (ROC) curve is commonly used for evaluating the performance of detectors at different operating points, without committing to a single decision threshold [23]. ROC analysis is robust to imprecise class distributions and misclassification costs [24]. In a previous work, the authors proposed a ROC-based iterative Boolean combination (*IBC*) technique for fusion of responses from any crisp or soft detector trained on *fixed-size* data sets [25]. The *IBC* algorithm inputs *all* generated classifiers, and combines their responses in the ROC space by applying all Boolean functions, over several iterations until the ROC convex hull (ROCCH) stops improving. *IBC* was applied to combine the responses from a multiple-HMM ADS, where each HMM is trained through batch learning using the *same* data but with different number of states and initializations. Results have shown that detection accuracy improves significantly, especially when training data are limited and class distributions are imbalanced [25]. The *IBC* is a general decision-level fusion technique in the ROC space that aims at improving ensembles accuracy, when learning is performed from a fixed-size set of limited training data. However, *IBC* does not allow to efficiently adapt a fusion function over time when new data becomes available, since it requires a fixed number of classifiers.

In contrast, this paper presents a new ROC-based system to efficiently adapt EoHMMs over time, from new training data, according to a learn-and-combine approach. Given new training data, a new pool of HMMs is generated from newly acquired data using different HMM states and initializations. The responses from these newly trained HMMs are then combined with those of the previously trained HMMs in ROC space using the *incremental* Boolean combination (*incrBC*) technique. *IncrBC* extends on *IBC* in that Boolean fusion of HMMs may be adapted for new data, without multiple iterations. However, system efficiency and scalability remain a serious issue. On its own, *incrBC* allows to discard previously learned data, yet it retains all previously generated HMMs in the pool. Over time, the pool size would therefore grow indefinitely, yielding unnecessarily high memory requirements, and increasingly complex Boolean fusion rules. Operating ever-growing EoHMMs would increase computational and memory complexity.

To overcome these limitations, specialized algorithms are proposed for memory management. First, ensemble selection algorithms are proposed to efficiently select diversified EoHMMs from the pool, and to adapt the Boolean fusion functions and decision thresholds to improve overall system performance. The proposed system may employ one of the two ensemble selection algorithms, called BC_{greedy} and BC_{search}, that are adapted to benefit from the monotonicity in *incrBC* accuracy, for a reduced complexity. Both algorithms feature rank- and search-based selection strategies to optimize ROOCH of combinations, and hence maximize the area under the ROCCH (AUCH). Both algorithms preserve the ROCCH of combination which allows visualizing the expected performance over the entire ROC space, and adapting to changes in operating conditions during system operations, such as tolerated false alarm rate, prior probabilities and costs of errors. Most existing techniques for adapting ensembles of classifiers require restarting the training, selection, combination, and optimization procedures to account for such a change in operating conditions. Finally, to address the potentially growing memory requirements over time, the proposed system integrates a memory management strategy to prune less relevant HMMs from the pool.

For a proof-of-concept validation, this system is applied to adaptive anomaly detection from system call sequences. The experiments are conducted on both synthetically generated data and sendmail data from the University of New Mexico (UNM) data sets[1] [1]. UNM data sets are still commonly used in literature [26,27] due to limited publicly available system call data sets. In fact labeling real system call sequences is a challenging task, which depends on detector window size. To overcome issues encountered when using real-world system call data for anomaly based HIDS, the experimental results are complemented with synthetically generated data. The synthetic data generator is based on the Conditional Relative Entropy (*CRE*), and is closely related to the work of Tan and Maxion [28]. It allows simulating different processes with various complexities and provides the desired amount of normal data for training and labeled data (normal and anomalous) for testing. For both real and synthetic data sets, the data are organized into several blocks that are assumed to have been acquired over time. The new data allow to account for rare events, and adapt to changes in monitored environments such as application updates or changes in user behavior over time. The performance of the proposed system is compared to that of the reference batch BW (BBW) trained on all cumulative data, of on-line BW (OBW) [9], and of an algorithm for incremental learning of HMM parameters based on BW (IBW) [10]. The performance of the median and majority vote fusion functions, combining outputs from the

[1] http://www.cs.unm.edu/~immsec/systemcalls.htm.

previously generated pool of HMMs, are also presented. Accuracy is assessed using the area under the ROC curve (AUC) [29] and the true positive rate (*tpr*) achieved at a fixed false positive rate (*fpr*) value.

The rest of this paper is organized as follows. The next section briefly reviews the HMM, as it applies to adaptive ADS from system call sequences. It also reviews techniques for incremental learning of HMM parameters, and for designing multiple classifiers systems. Section 3 describes the proposed ROC-based system for efficient adaptation of EoHMM from new data according to the learn-and-combine approach, including new techniques for Boolean combination and model management. Section 4 presents the experimental methodology (data sets, evaluation methods and performance metrics) used for proof-of-concept computer simulations. Simulation results are presented and discussed in Section 5.

2. Adaptive anomaly detection systems

2.1. Anomaly detection using HMMs

HMMs have been shown successful in detecting anomalies in sequences of operating system calls [1,2,28,30,31]. A discrete-time finite-state HMM is a stochastic process determined by two interrelated mechanisms—a latent Markov chain having a finite number of states N, and a set of observation probability distributions, each one associated with a state. Starting from an initial state $S_i \in \{S_1, \ldots, S_N\}$, determined by the initial state probability distribution π_i, at each discrete-time instant, the process transits from state S_i to state S_j according to the transition probability distribution a_{ij}. The process then emits a symbol v_k, from a finite alphabet $V = \{v_1, \ldots, v_M\}$ of size M symbols, according to the discrete-output probability distribution $b_j(v_k)$ of the current state S_j. HMM is commonly parametrized by $\lambda = (\pi, A, B)$, where the vector $\pi = \{\pi_i\}$ is the initial state probability distribution, matrix $A = \{a_{ij}\}$ is the state transition probability distribution, and matrix $B = \{b_j(v_k)\}$ is the state output probability distribution, ($1 \le i,j \le N$ and $1 \le k \le M$).

HMMs can perform three canonical functions, evaluation, decoding and learning [32]. Evaluation aims to compute the likelihood of an observation sequence $o_{1:T}$ given a trained model λ, $P(o_{1:T}|\lambda)$. The likelihood is typically evaluated by using a fixed-interval smoothing algorithm such as the Forward–Backward (FB) [32] or the Forward–Filtering Backward-Smoothing (FFBS) [33]. Decoding means finding the most likely state sequence S that best explains a given observation sequence $o_{1:T}$, i.e., maximize $P(S|o_{1:T}, \lambda)$. The best state sequence is commonly determined by the Viterbi algorithm. Learning aims to estimate HMM parameters λ to best fit the observed batch of data $o_{1:T}$. Typically, parameters estimation consists of maximizing the likelihood of the training data over HMM parameters space, $P(o_{1:T}|\lambda)$. Since this likelihood depends on the latent states, there is no known analytical solution to the learning problem. Iterative optimization techniques such as the Baum–Welch (BW) algorithm [7]—a special case of the Expectation-Maximization (EM) [34], or standard numerical optimization methods such as gradient descent [6,35] are often employed in practice. In either case, HMM parameters are estimated over several training iterations, until the likelihood function is maximized over data samples. Each iteration typically involves observing all available training data to evaluate the likelihood value and estimate the state densities by using a fixed-interval smoothing algorithm.

Anomaly detection creates a profile that describes normal behaviors. Events that deviate significantly from this profile are considered anomalous. In computer and network security, legitimate system call sequences generated during the normal execution of a privileged process are typically employed in host-based ADSs [1,3]. As stochastic models for temporal data, HMMs provide compact detectors that are able to capture probability distributions corresponding to complex real-world phenomena, with tolerance to noise and uncertainty. Given a sufficiently large training set of legitimate system call observations, an ergodic[2] HMM trained according to the BW or gradient-based algorithms, is able to capture the underlying structure of the monitored process [1,2]. During operations, system calls sequences generated by the monitored process are evaluated with the trained HMM, according to the FB or FFBS algorithms, which should assign significantly lower likelihood values to anomalous sequences than to normal ones. By setting a decision threshold on the likelihood values, monitored sequences can be classified as normal or anomalous.

Estimating the parameters of a HMM requires the specification of the number of hidden states. Although it is a critical parameter for HMM performance, this number is often chosen heuristically or empirically, by selecting the single value that provides the best performance on training data [1,2,31]. Using a single HMM with a pre-specified number of states may have limited capabilities to capture the underlying structure of the data. HMMs trained with a fixed-size training data and with a different number of states and initializations have been shown to provided a higher level of performance in host-based ADSs [30].

2.2. Adaptation in anomaly detection

The design of accurate detectors for ADS requires the collection of a sufficient amount of representative data. In practice however, it is very difficult to collect, analyze and label comprehensive data sets for reasons that range from technical to ethical. Limited normal data are typically provided for training HMM-based detectors, and limited labeled data are also provided for validation of an ADS. Furthermore, the underlying data distribution of normal behaviors may vary according to changes in the monitored environment such as an application update or changes in users behavior. The ability to incrementally adapt HMM parameters in response to newly acquired training data from the operational environment or other sources is therefore an undisputed asset for sustaining a high level of performance.

Assume that an HMM-based ADS is trained using a finite set of system call data. During operations, monitoring a centralized server, for instance, the system is susceptible to produce a higher false alarms rate than tolerated by the system administrator. This is largely due to the limited amount of representative training data. The ADS will have an incomplete view of the normal process behavior, and rare events will be mistakenly considered as anomalous. The HMM detector should be refined incrementally over time using some additional normal data, to better fit the normal behavior of process in consideration. Such situations may also occur when an ADS specialized for monitoring a specific process is implemented on different hosts machines with different settings and functionality.

[2] An HMM with an ergodic or fully connected topology is typically employed in situations where the hidden states have no physical meaning such as modeling a process behavior using its generated system calls.

Generic models that are trained using data corresponding to common settings, could be set into operation then incrementally refined to fit the normal process behavior on each host. Otherwise, training data must be collected and analyzed from the same process on every host.

As a part of an ADS, the system administrator plays a crucial role in providing new data for training and validation of the detectors. When an alarm is raised, the suspicious system call sub-sequences are logged and analyzed for evidence of an attack. If an intrusion attempt is confirmed, the corresponding anomalous system calls should be provided to update the validation set ($\mathcal{V}$). Otherwise, the intrusion attempt is considered as a false alarm and these rare sub-sequences are tagged as normal and employed to update the HMM detectors. One challenge is the efficient integration of the newly acquired data into the ADS without corrupting the existing knowledge structure, and thereby degrading the performance.

2.3. Techniques for incremental learning of HMM parameters

Incremental learning refers to the ability of an algorithm to learn from new data after a classifier has already been trained from previous data and deployed for operations. As illustrated in Fig. 1, once a new block of data becomes available, incremental learning produces a new hypothesis (i.e., decision rule) that depends only on the previous hypothesis and the current training data [5]. Incremental learning allows to decrease the memory requirements needed to learn the new data, since there is no need to store and access previously learned data. Furthermore, since training is performed only on new training blocks, and not on all accumulated data, incremental learning would also lower the time complexity needed to learn new data.

The main challenge of incremental learning is the ability to sustain a high level of performance without corrupting previously learned knowledge [4,5]. In fact, considering the HMM parameters obtained using a batch learning technique on a first block of sub-sequences as starting point for training the HMM on a second newly acquired block, may not allow the optimization process to escape the local maximum associated with the first block. Remaining trapped in local maxima leads to knowledge corruption and hence to a decline in system performance (see Fig. 2).

Several on-line learning techniques have been proposed in literature to re-estimate HMM parameters from a continuous stream of symbols. The objective is to optimize HMM parameters to fit the source generating the data through one observation of the data. They are typically designed to reduce the training time and memory complexity and to accelerate the convergence when learning long streams of data. These techniques are derived from their batch counterparts, and include EM-based [9,8] and gradient-based [35,36] techniques. However, the on-line optimization and update of HMM parameters are continuously performed upon observing each new sub-sequence [9,35] or new symbol [8,36] of observation, with no iterations.

In practice however, when an on-line learning technique is applied to incremental learning of a finite data set, one pass over the data block is insufficient to capture the phenomena. In addition, the convergence properties of these on-line algorithms no longer hold when

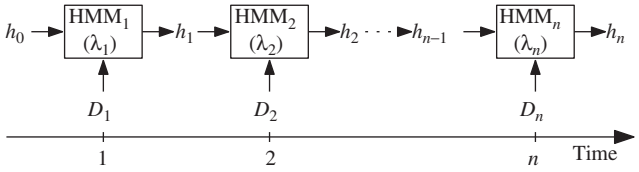


Fig. 1. An incremental learning scenario where blocks of data are used to update the HMM parameters (λ) over a period of time. Let $D_1, D_2, \ldots, D_n$ be the blocks of training data available to the HMM at discrete instants in time. The HMM starts with an initial hypothesis h_0 associated with the initial set of parameters λ_0, which constitutes the prior knowledge of the domain. Thus, h_0 and λ_0 get updated to h_1 and λ_1 on the basis of D_1, then h_1 and λ_1 get updated to h_2 and λ_2 on the basis of D_2, and so forth.

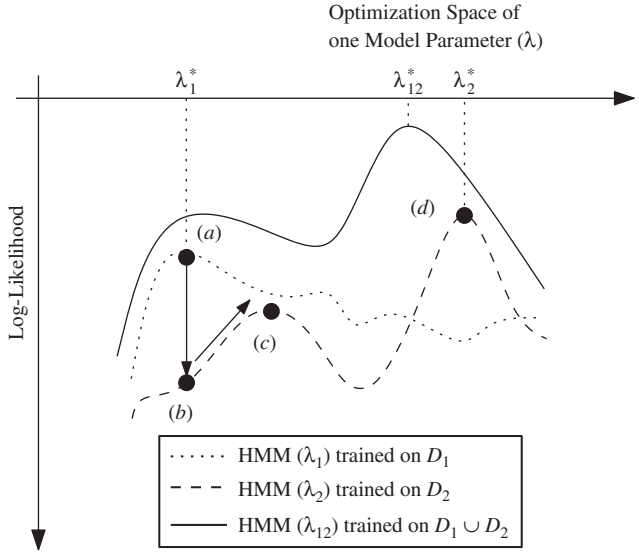


Fig. 2. An illustration of the decline in performance that may occur with batch or on-line estimation of HMM parameters, when learning is performed incrementally on successive blocks.

provided with limited amount of data. Several iterations over the new block of observations are therefore required to better fit the HMM parameters to the newly acquired data. When several iterations are allowed, on-line algorithms may attain a local maximum on the log-likelihood function associated with the current block of data, and must overcome the same issue of remaining trapped in the previous local maximum (see Fig. 2).

Fig. 2 illustrates the decline in performance that may occur with batch or on-line algorithms during incremental update of one HMM parameter. The values λ_1^*, λ_2^*, and λ_{12}^* correspond to the optimal values of parameters after learning D_1,D_2 and $D_1 \cup D_2$, respectively. Assume that the training block D_2, which contains one or multiple sub-sequences, becomes available after an HMM(λ_1) has been previously trained on a block D_1 and deployed for operations. An incremental algorithm that optimizes, for instance, the log-likelihood function must re-estimate the HMM parameters over several iterations until this function is maximized for D_2. The optimization of HMM parameters depends on the shape and position of the log-likelihood function of HMM(λ_2) with respect to that of HMM(λ_1). When an incremental learning technique is employed to train on D_2 alone, λ_1^* is the starting point for re-estimating the HMM. If the log-likelihood function of HMM(λ_2) has some local maxima in the proximity of λ_1^*, the optimization may remain trapped in these maxima, and hence does not accommodate D_2, providing a similar model parameter. If λ_1^* was selected according to point (a), the optimization will become trapped in the local maximum at point (c) instead of approaching λ_{12}^*. This leads to a knowledge corruption, and a decline in HMM performance. In contrast, training a new HMM(λ_2) on D_2 from the start using different initializations may lead to point (d). As described in subsequent sections, combination of responses from two HMMs selected according to λ_1^* and λ_2^* exploits their complementary information for a higher overall level of performance.

2.4. Incremental learning with ensembles of classifiers

Multiple classifier systems (MCSs) are based on the combination of several accurate and diverse base classifiers to improve the overall system accuracy [11,12,15,16]. MCSs have been employed to overcome the limitations faced with a single classifier system [11–14]. Classifiers may converge to different local optima according to different initializations or different parameter values. Different classifiers can have different domains of expertise or perceptions of the same problem. Therefore, combining the predictions of an ensemble of classifiers (EoCs) reduces the risk of selecting a single classifier with a poor generalization performance. Furthermore, base classifiers may commit different errors providing complementary information that increase the ensemble accuracy. EoCs provide alternative solutions for incremental learning by combining classifiers trained on each block of data that becomes available, instead of selecting and updating a single classifier.

In general, the design of EoCs involves generating a diversified pool of base classifiers, selecting a subset of members from that pool, and then combining their output predictions such that the overall accuracy and reliability is improved. A pool can be composed of either homogeneous or heterogeneous classifiers. Homogeneous classifiers are generated by the same classifier trained using different manipulations of parameters, training data [18,19], or input features [37]. Heterogeneous classifiers are generated by training different classifiers on the same data set.

The combination of selected classifiers typically occurs at the score, rank or decision levels [12,14], and may be static (e.g., sum, product, majority voting), adaptive (e.g., weighted average, weighted voting) or trainable (also known as stacked or meta-classifier). Fusion at the score level is most prevalent in literature [16]. Normalization of the scores is typically required before applying static or adaptive fusion functions, which may not be a trivial task for heterogeneous classifiers. Trainable approaches, where a global meta-classifier is trained for fusion on classifiers responses [16,38–40], are prone to overfitting and require a large independent validation set for estimating the parameters of the combining classifier [38,39]. Fusion at the rank level is mostly suitable for multi-class classification problems, where the correct class is expected to appear in the top of the ranked list [20,41]. Fusion at the decision level exploits the least amount of information since only class labels are considered. Compared to the other fusion methods, it is less investigated in literature. Majority voting [42], weighted majority voting [43,44] and behavior-knowledge-space (BKS) methods [45] are the most representative decision-level fusing methods. One issue that appears with decision level fusion is the possibility of ties. BKS only applies to low dimensional problems, and requires a large independent training and validation set to provide accurate combination rules.

EoCs may be adapted for incremental learning by generating a new pool of classifiers as each block of data becomes available, and combining the outputs with those of previously generated classifiers with some fusion technique [46]. The main advantage of using EoCs to implement a learn-and-combine approach is the possibility of avoiding knowledge corruption as classifiers are trained from the start on a new block of data.

For instance, the adaptive boosting methods employed in AdaBoost [19] have been extended for incremental learning from new blocks of data in Learn++ [5]. Given a fixed-size training set, AdaBoost iteratively generates a sequence of weak classifiers each focusing on training observations that have been misclassified by the previous classifier. This is achieved by updating a weight distribution over the entire training set according to the performance of the previously generated classifier. At each iteration, AdaBoost increases the weights of wrongly classified training observations and decreases those of correctly classified observations, such that the new classifier focuses on hard-to-classify observations. The final output is a weighted majority voting of all generated classifiers. By contrast, Learn++ generates a number of base classifiers for each block of data that becomes available. The weight distribution is updated according to the performance of all previously generated classifiers, which allows to accommodate new classes [5]. The outputs are then combined through weighted majority voting to obtain the final classification. Improvements proposed for this algorithm differ by the combination functions and weight distribution update. In particular, Learn++ variant for imbalanced data employs a class-conditional weight factor for updating classifiers weight [47].

An on-line version of bagging and boosting ensembles have been proposed for learning from labeled streams of data [48]. The bootstrap sampling and weight distribution update that are employed for fixed-sized data sets are now simulated according to Poisson distribution for data streams. For on-line AdaBoost algorithm, the ensemble is composed of a fixed number of classifiers. Each new observation is presented sequentially to the ensemble members. A classifier is trained on this observation k times, where k is drawn from a Poisson distribution parametrized by the observation weight. When an observation is misclassified its weight increases and hence will be presented more often to the following classifier in the ensemble. The final output is the weighted majority vote over all the base classifiers. However, on-line bagging and boosting techniques consider learning continuously from labeled streams of data.

The techniques described above are designed for two- or multi-class classification and hence require labeled training data sets to compute the errors committed by the base classifiers and update the weight distribution at each iteration. In HMM-based ADSs, the HMM detector is a one-class classifier that is trained using the *normal* system call data only as described in Sections 2.1 and 2.2.. Therefore, they are not suitable for ADSs using system call sequences. Some authors however, adopt these techniques by considering that rare normal system call sequences are anomalous. For instance, an EoHMMs based on discrete AdaBoost and its on-line version has been applied for anomaly detection [31]. With this method, an arbitrary threshold is set according to the log-likelihood output from HMMs trained on the normal system call behavior, below which normal system call sequences are considered as anomalous. Five HMMs are trained with the same number of states on fixed-sized data, and then employed as base detectors in AdaBoost algorithm. When new data becomes available, these HMMs are updated according to an on-line AdaBoost variant [48]. Threshold setting is shown to have a significant impact on the detection performance of the EoHMMs [31]. In fact, rare system call events are normal, if they are considered anomalous during the design phase, they will generate false alarms during the testing phase. These rare events may be suspicious if, during operation, they occur in bursts over a short period of time.

Another specific combination technique for HMMs consists of weight averaging the parameters of an HMM trained on a newly acquired block of data with those of a previously trained HMMs [2]. Although this technique may work for left-right HMMs, it is not suitable for ergodic HMMs as their states are permuted with each different training phase, and hence averaging parameters leads to knowledge corruption. Furthermore, it is restricted to combining HMMs with the same number of states.

For one- or two-class classification problems, Boolean combination of classifier responses in the ROC space provides several advantages over related fusion techniques [25,49]. It does not require any prior assumption regarding the independence of classifiers, in contrast with other fusion functions such as sum, product or averaging that are based on the independence of classifiers [16,22]. In addition, normalization of heterogeneous classifiers output scores is not required, because ROC curves are invariant to monotonic transformation of classification thresholds [23]. Boolean combination techniques may therefore be applied to combine the responses from any crisp or soft homogeneous or heterogeneous classifiers, or from classifiers trained on different data. Most existing ensemble techniques, especially those proposed for incremental learning, aim at maximizing the system performance through a single measure of accuracy. This implicitly assumes fixed prior probabilities and misclassification costs. Furthermore, an arbitrary and fixed threshold is typically set (implicitly or explicitly) on the output scores of soft classifiers of an ensemble prior to taking a majority voting decision. In anomaly detection however, prior class distributions are highly imbalanced and misclassification costs are different and both may change over time. As detailed in Section 3.1, Boolean combination in the ROC space aims at maximizing the overall convex hull of combinations overall classifiers decision thresholds, which allows to adapt to changes in prior probabilities and cost of errors during system operation.

3. Learn-and-combine approach using incremental boolean combination

In this section, a ROC-based system is proposed to efficiently adapt EoHMMs over time, from new training data, according to a learn-and-combine approach. As new blocks of data become available, the system trains new HMMs to evolve a pool of classifiers, and performs incremental Boolean combination (*incrBC*) of a diversified subset of HMM responses in the ROC space. This is achieved by selecting the HMMs, decision thresholds, and Boolean functions such that the overall EoHMMs performance is maximized.

The main components of the system are illustrated in Fig. 3 and described in Algorithm 1. When a new block of *normal* training sub-sequences (D_k) becomes available, a new pool of base HMMs ($\mathcal{P}_k$) is generated according to different number of states and random

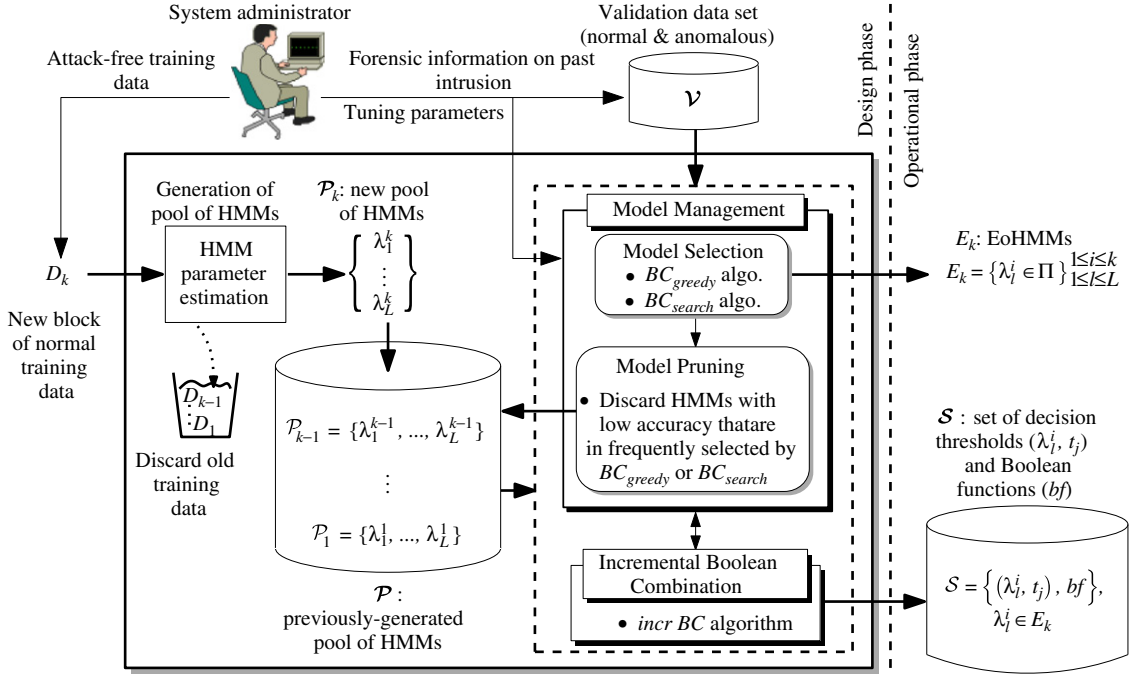


Fig. 3. Block diagram of the adaptive system proposed for *incrBC* of HMMs, trained from newly acquired blocks of data D_k, according to the learn-and-combine approach. It allows for an efficient management of the pool of HMMs and selection of EoHMMs, decision thresholds, and Boolean functions.

initializations. (Further details on HMMs training and validation are given in Section 4.2.) $\mathcal{P}_k$ is then appended to the previously generated pool of base HMMs $\mathcal{P}=\{\mathcal{P}_1,\mathcal{P}_2,\ldots,\mathcal{P}_{k-1}\}\cup\mathcal{P}_k$, and D_k is discarded. The model management module, described in Section 3.2, performs model selection and pruning. The selection module forms the EoHMMs (E_k) considered for operations, by selecting HMMs from the pool $\mathcal{P}$ that most improve the overall system performance according to one of two proposed model selection algorithms, BC_{greedy} and BC_{search} (see Algorithms 3 and 4), both of which depend on the incremental Boolean combination module. The pruning module controls the size of the pool by discarding less accurate HMMs that have not been selected for some user-defined time interval. To form an EoHMMs, the incremental Boolean combination module combines the responses from the selected HMMs in the ROC space using all Boolean functions according to the *incrBC* algorithm (see Algorithm 2 in Section 3.1). The *incrBC* algorithm also selects and stores the set ($\mathcal{S}$) of decision thresholds (from each base HMM of the EoHMMs) and Boolean functions that most improves the overall EoHMMs performance. The HMMs that form the EoHMMs, E_k, and the set of decision thresholds and Boolean functions, $\mathcal{S}$, are used during operations.

Algorithm 1. Learn-and-combine approach using incremental Boolean combination.

input: Training data blocks $D_1,D_2,\ldots,D_K$ that become available over time (normal sub-sequences)
Labeled validation data set $\mathcal{V}$ (normal and anomalous sub-sequences)
output: Selected HMMs to form the EoHMMs (E_k) of size $|E_k|$
Selected set ($\mathcal{S}$) of decision thresholds (λ_l^i,t_j) and Boolean functions, each of size 2 to $|E_k|$ detector

select $selection\_algo \in \{BC_{greedy},BC_{search}\}$; //choose the model selection algorithm
set $count_i \leftarrow 0, \forall \lambda_l^i \in \mathcal{P}$; //counter for unselected HMMs over time
set LT; //lifetime expectancy of models in the pool
foreach D_k **do**
  train new pool of HMMs, $\mathcal{P}_k=\{\lambda_1^k,\ldots,\lambda_L^k\}$; // use different no. states and initializations
  $\mathcal{P}\leftarrow\mathcal{P}\cup\mathcal{P}_k$; // add the new pool to previously learned pools
  switch *selection_algo* **do**
    case BC_{greedy}
    $E_k = BC_{greedy}(\mathcal{P},\mathcal{V})$; // use greedy selection (Algorithm 3)
    case BC_{search}
    $E_k = BC_{search}(\mathcal{P},\mathcal{V})$; // use incremental search (Algorithm 4)
    // both algorithms call the incrBC algorithm(Algorithm 2) for selecting and storing the set of best decision thresholds and Boolean functions, $\mathcal{S}=\{(\lambda_l^i,t_j),bf\}$, $\lambda_l^i \in E_k$
    $count(\lambda_l^i)\leftarrow count(\lambda_l^i)+1, \forall \lambda_l^i \in \mathcal{P}\backslash E_k$; // increment counter for unselected HMMs
  if $k > LT$ **then**
    prune $\lambda_l^i : count(\lambda_l^i) > LT$; // discard HMMs that are not selected for LT blocks from $\mathcal{P}$
return E_k and $\mathcal{S}$

A set of validation data ($\mathcal{V}$) composed of *normal* and *anomalous* sub-sequences is required during the design phase for selection of HMMs, decision thresholds, and Boolean functions. It is managed and input by the system administrator as discussed in Section 2.2. When manifestations of novel attacks are discovered, relevant anomalous sub-sequences are then stored to update the validation set. The HMMs, decision thresholds, and Boolean combinations must then be re-selected to accommodate the new information. Furthermore, the system administrator ensures that the blocks of training data provided for updating the pool of HMMs are clean from intrusions. He is also responsible for selecting and tuning the model management algorithms to trade-off the size of the pool and EoHMMs against overall system accuracy.

The proposed approach inherits all desirable properties of Boolean combination in the ROC space as detailed in the following sections. It covers the whole performance range of false and true positive rates, which allows for a flexible selection of the desired operating performance. As conditions change, such as prior probabilities and costs of errors, the overall convex hull of combinations does not change; only the portion of interest. This change shifts the optimal operating point to another vertex on the composite convex hull. The corresponding HMMs are then activated, and their responses are combined according to different decision thresholds and Boolean functions.

3.1. Incremental boolean combination in the ROC space

The incremental Boolean combination (*incrBC*) technique summarized in this section has been proposed by the authors [25,50] to combine the responses from models trained with different number of states and initialization on a fixed-size data set. In the learn-and-combine approach for incremental learning, it is adapted to combine the responses of HMMs trained with different number of states and initialization, however using new data blocks that are acquired over time.

A *crisp* detector outputs a class label while a *soft* detector assigns scores or probabilities to the input samples, which can be converted to a crisp detector by setting a decision threshold on the scores. Given the responses of a crisp detector on a validation set, the true positive rate (*tpr*) is the proportion of positives correctly classified over the total number of positive samples. The false positive rate (*fpr*) is the proportion of negatives incorrectly classified over the total number of negative samples. A ROC curve is a plot of *tpr* against *fpr*. A crisp detector produces a single data point in the ROC plane, while a soft detector produces a ROC curve by varying the decision thresholds. For equal priors and cost of errors, the optimal decision threshold (minimizing overall costs) corresponds to the vertex that is closest to the upper-left corner of the ROC plane. Given two operating points, say a and b, in the ROC space, a is defined as *superior* to b if $fpr_a \le fpr_b$ and $tpr_a \ge tpr_b$. If one ROC curve has all its points superior to those of another curve, it *dominates* the latter. If a ROC curve has $tpr_x > fpr_x$ for all its points x then, it is a *proper* ROC curve.

In practice, the number of decision thresholds is the number of distinct score values assigned by a soft detector to the validation samples. This number also corresponds to the number of resulting crisp detectors and to the number of vertices on the empirical ROC plot. In fact, an empirical ROC plot is obtained by connecting the observed (*tpr*,*fpr*) pairs of a soft detector at each decision threshold. With a finite number of decision thresholds, an empirical ROC plot is a step-like function which approaches a true curve as the number of samples, and hence the number of decision thresholds, approaches infinity. Therefore, it is not necessarily convex and proper. Concavities indicate local performance that is worse than random behavior and may provide diverse information.

The convex hull of an empirical ROC plot (ROCCH) is the smallest convex set containing its vertices, i.e., the piece-wise outer envelope connecting only its superior points. It may be used to combining detectors based on a simple interpolation between their responses [24,51]. This approach has been called the maximum realizable ROC (MRROC) [51] since it represents a system that is equal to, or better than, all the existing systems for all Neyman–Pearson criteria [52]. However, the MRROC discards responses from inferior detectors which may provide diverse information for an improved performance. Using Boolean conjunction and disjunction functions to combine the responses of multiple soft detectors in the ROC space have shown and improved performance over the MRROC [49,53]. However, these techniques assume that the detectors are conditionally independent, and their of ROC curves are convex. These assumptions are violated in most real-world applications, especially when detectors are designed using limited and imbalanced training data. Furthermore, the correlation between soft detector decisions also depends on the threshold selections.

The *incrBC* technique applies *all* Boolean functions to combine the responses of multiple crisp or soft detectors, requires no prior assumptions, and selects the thresholds and Boolean functions that improve the MRROC [25]. By applying all Boolean functions to combine the responses of detectors at each corresponding decision threshold, the *incrBC* technique implicitly accounts for the effects of correlation among detectors and accommodates for the concavities in the ROC curves. Indeed, AND and OR rules will not provide improvements for the inferior points that correspond to concavities. Other Boolean functions, for instance those that exploit negations of responses, may however emerge (see Fig. 4).

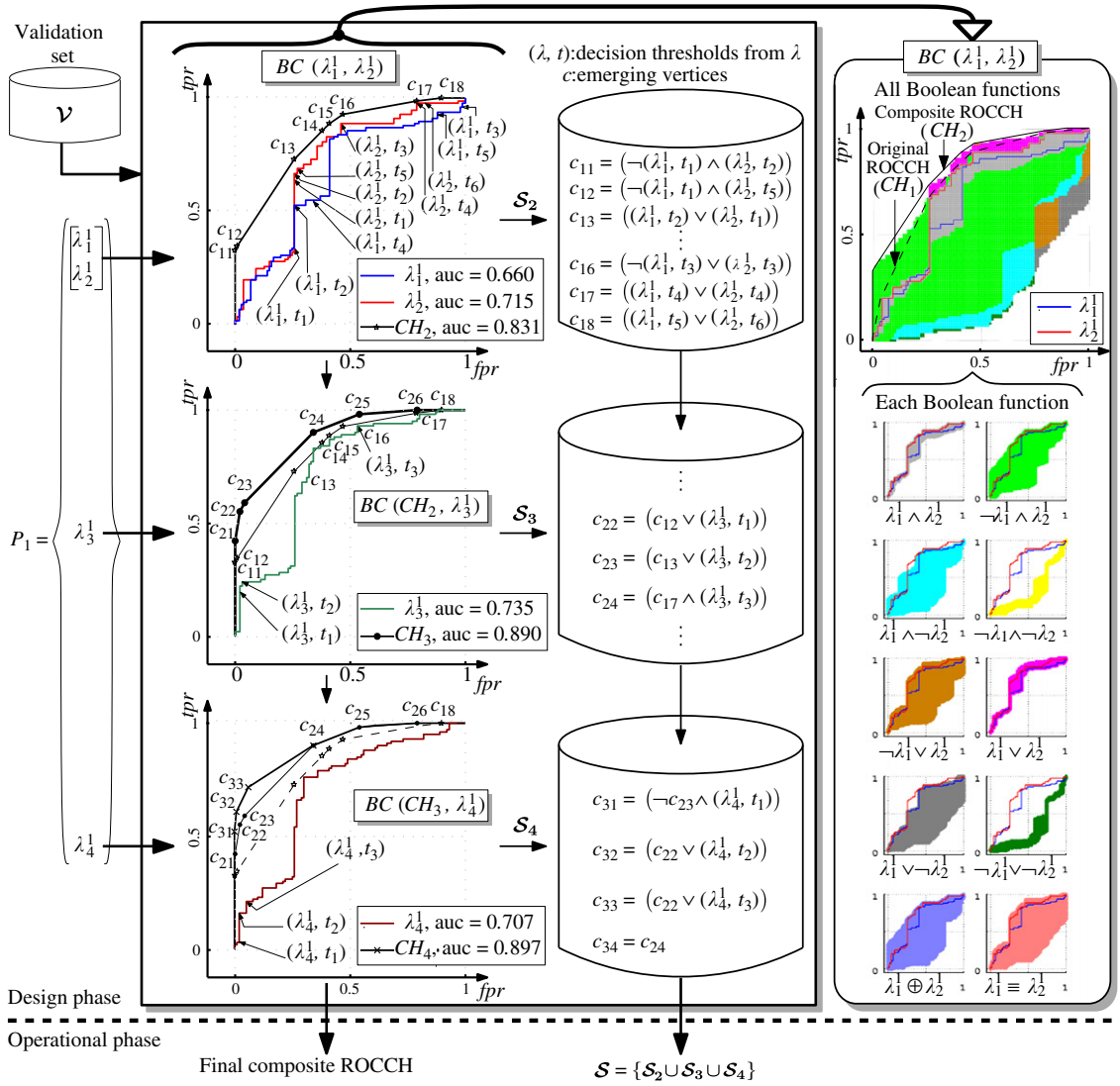


Fig. 4. An illustration of the steps involved during the design phase of the *incrBC* algorithm employed for incremental combination from a pool of four HMMs $\mathcal{P}_1 = \{\lambda_1^1, \ldots, \lambda_4^1\}$. Each HMM is trained with different number of states and initializations on a block (D_1) of normal data synthetically generated with $\Sigma = 8$ and $CRE = 0.3$ using the BW algorithm (see Section 4 for details on data generation and HMM training). At each step, the example illustrates the update of the composite ROCCH (*CH*) and the selection of the corresponding set ($\mathcal{S}$) of decision thresholds and Boolean functions for overall improved system performance.

Algorithm 2. *incrBC*($\lambda_1,\lambda_2,\ldots,\lambda_K,\mathcal{V}$): Incremental Boolean Combination of HMMs in ROC space.

input: K HMMs ($\lambda_1,\lambda_2,\ldots,\lambda_K$) and a validation set $\mathcal{V}$ of size $|\mathcal{V}|$
output: ROCCH of combined HMMs where each vertex is the result of 2 to K combination of crisp detectors. Each combination selects the best decision thresholds from different HMMs (λ_i,t_j) and the best Boolean function, which are stored in the set ($\mathcal{S}$)

$n_k \leftarrow$ no. decision thresholds of λ_k using $\mathcal{V}$; // no. vertices on ROC(λ_k) or no. crisp detectors
BooleanFunctions $\leftarrow \{a\wedge b,\neg a\wedge b,a\wedge\neg b,\neg(a\wedge b),a\vee b,\neg a\vee b,a\vee\neg b,\neg(a\vee b),a\oplus b,a\equiv b\}$
//combine responses from the first two HMMs
compute $ROCCH_1$ of thefirst two HMMs (λ_1 and λ_2)
allocate $\boldsymbol{F}$ an array of size: $[2,n_1\times n_2]$; // temporary storage of combination results
foreach $bf \in$ *BooleanFunctions* **do**
  for $i\leftarrow 1$ **to** n_1 **do**
    $\boldsymbol{R}_1 \leftarrow (\lambda_1,t_i)$; // responses of λ_1 at decision threshold t_i using $\mathcal{V}$
    for $j\leftarrow 1$**to** n_2 **do**
      $R_2 \leftarrow (\lambda_2,t_j)$; // responses of λ_2 at decision threshold $t_j$using $\mathcal{V}$
      $\boldsymbol{R_c} \leftarrow bf(\boldsymbol{R}_1,\boldsymbol{R}_2)$; // combine responses using current Boolean function
      compute(*tpr*,*fpr*) of $\boldsymbol{R_c}$ using$\mathcal{V}$; // map combined responses to ROC space (1 point)
      push (*tpr*,*fpr*)onto $\boldsymbol{F}$
  compute $ROCCH_2$ of all ROC points in $\boldsymbol{F}$
  $n_{ev} \leftarrow$number of emerging vertices; // no. vertices of $ROCCH_2$ superior to $ROCCH_1$
  $\mathcal{S}_2 \leftarrow \{(\lambda_1,t_i),(\lambda_2,t_j),bf\}$ // set of selected decision thresholds from each HMM and Boolean functions for emerging vertices
// combine responses of each successive HMM with the previously-combined responses
for $k\leftarrow 3$ **to** K **do**
  allocate $\boldsymbol{F}$ of size : $[2,n_k\times n_{ev}]$
  foreach $bf \in$ *BooleanFunctions* **do**
    for $i\leftarrow 1$ **to** n_{ev} **do**
      $\boldsymbol{R_i} \leftarrow S_{k-1}(i)$; // responses from previous combinations
      for $j\leftarrow 1$ **to** n_k **do**
        $\boldsymbol{R_k} \leftarrow (\lambda_k,t_j)$
        $\boldsymbol{R_c} \leftarrow bf(\boldsymbol{R_i},\boldsymbol{R_k})$
        compute (*tpr*,*fpr*)of $\boldsymbol{R_c}$ using $\mathcal{V}$
        push (*tpr*,*fpr*)onto $\boldsymbol{F}$
  compute $ROCCH_k$ *of all ROC points in* $\boldsymbol{F}$
  $n_{ev} \leftarrow$number of emerging vertices; // no. vertices of $ROCCH_k$ superior to $ROCCH_{k-1}$
  $\mathcal{S}_k \leftarrow \{\mathcal{S}_{k-1}(i),(\lambda_k,t_j),bf\}$ // set of selected subset from previous combinations, decision thresholds from the newly selected HMM, and Boolean functions for emerging vertices
store $\mathcal{S}_k, 2\le k\le K$
return $ROCCH_K$

The main steps of *incrBC* are presented in Algorithm 2. Given a pool of K HMMs $\mathcal{P}=\{\lambda_1,\lambda_2,\ldots,\lambda_K\}$ having n_k, ($k=1,\ldots,K$), distinct decision thresholds on a validation set ($\mathcal{V}$), the *incrBC* algorithm starts by combining the responses of the first two HMMs in the pool. Each of the ten Boolean functions is applied to combine the responses of each decision threshold from the first HMM (λ_1,t_i), $i=1,\ldots,n_1$, with the responses of each decision threshold from the second HMM (λ_2,t_j), $j=1,\ldots,n_2$. The fused responses are then mapped to points (*fpr*,*tpr*) in the ROC space and their ROCCH is computed. Then, *incrBC* selects the emerging vertices which are superior to the ROCCH of original HMMs, stores the set ($\mathcal{S}$) of selected decision thresholds from each HMM and Boolean functions, and updates the ROCCH to include the new emerging vertices. The responses of previously emerging vertices, resulting from the first two HMMs, are then combined with the responses of the third HMM and so on, until the last HMM. The *incrBC* algorithm outputs the final composite ROCCH for visualization and selection of operating points (see Fig. 4). It also stores the selected set of decision thresholds and Boolean functions ($\mathcal{S}=\{(\lambda_i,t_j),bf\}, 2\le i\le K; 1\le j\le n_i$), for each vertex on the composite ROCCH to be applied during operations.

During the operation phases applied to incremental learning, the system uses one vertex or the interpolation between a pair of vertices on the facets of the final composite ROCCH, depending on the desired false alarm rate. Given a tolerable *fpr* value, the corresponding vertex or pair of vertices (which form an edge intersecting the vertical line at the specified *fpr* value) are first located on the ROCCH. The decision thresholds and Boolean functions, which have been derived and stored during the design phase, are then extracted and applied during the operational phase (see Fig. 5). When the operational conditions change, the optimal operating point can be shifted during operations to adapt for the changes by activating a different set of decision thresholds and Boolean functions. If, for instance, c_{32} (in Fig. 5) has become the optimal operating point because of a different cost of errors or a different tolerance in *fpr*, then c_{op} can be shifted during operation to $c_{32}=(((\neg(\lambda_1^1,t_1)\wedge(\lambda_2^1,t_5))\vee(\lambda_3^1,t_1))\vee(\lambda_4^1,t_2))$. The activated decision thresholds and Boolean functions are then updated accordingly.

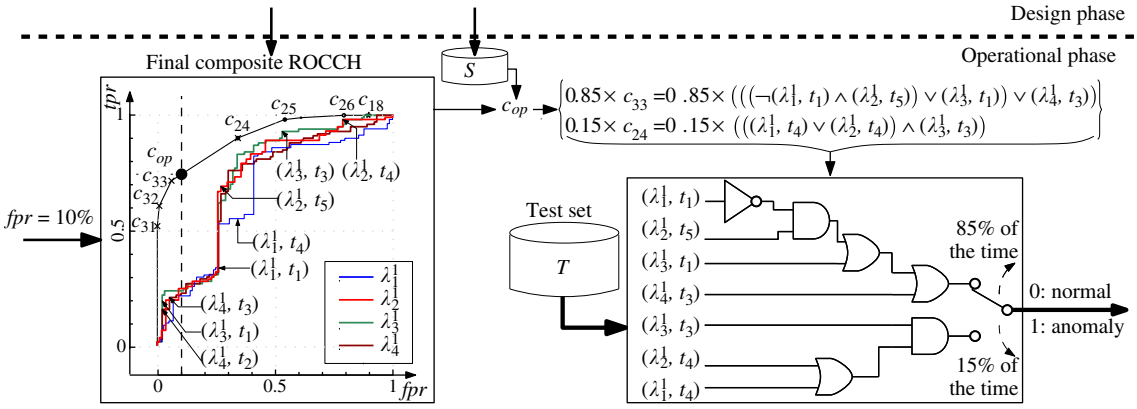


Fig. 5. An illustration of the *incrBC* algorithm during the operational phase. The example presents the set of decision thresholds and Boolean functions that are activated given, for instance, a maximum specified *fpr* of 10% for the system designed in Fig. 4. The operational point (c_{op}) that corresponds to $fpr=10\%$ is located between the vertices c_{33} and c_{24} of the composite ROCCH, which can be achieved by interpolation of responses [24,51]. The desired c_{op} is therefore realized by randomly taking the responses from c_{33} with probability value of 0.85 and from c_{24} with probability value of 0.15. The decision thresholds and Boolean functions of c_{33} and c_{24} are then retrieved from $\mathcal{S}$ and applied for operations.

During the design phase, the worst-case time and memory complexity for *incrBC* of a pool of K HMMs are $\mathcal{O}(Kn_1n_2)$ Boolean operations and $\mathcal{O}(n_1n_2)$ floating point registers, where n_k is the number of distinct decision thresholds of λ_k on a validation set $\mathcal{V}$. They can be roughly stated as functions of n_1 and n_2, because the number of emerging vertices (n_{ev}) from each successive combination of the remaining HMMs is typically lower than n_1 and n_2. During operations, the computational overhead of the activated set of Boolean functions is lower than that of operating the required number of HMMs. Therefore, the worst-case time and memory complexity is limited to operating the K HMMs.

By selecting crisp detectors from *all* available HMMs in the pool, the *incrBC* algorithm yields to unnecessarily high computational and storage cost when the pool size grows as new blocks of data become available. In addition, HMMs are combined according to the order in which they are stored in the pool. The *incrBC* algorithm is sensitive to the order in which the HMMs are selected and presented for combination. In fact, an HMM trained on new data block may capture different underlying data structure and could replace several previously selected HMMs. Model management strategies are therefore required to manage system resources, and overcome these limitations.

3.2. Model management

When batch learning is performed on a fixed-size data set, a fixed number of classifiers is typically generated. In this case, model management consists in selecting the most accurate and diverse base classifiers from the pool and, if necessary, pruning less relevant classifiers from the pool to reduce computational costs. The selected classifiers form an ensemble to be applied during system operations. Therefore, ensemble selection and pruning are commonly used interchangeably in related literature [21]. When learning is performed incrementally, the pool size grows as new blocks of data become available over time. Different ensemble selection and pruning mechanisms are therefore required. Ensemble selection is performed at each learning stage, when a new pool of classifiers is generated from a new block of data, and selected classifiers are deployed for operations. It involves applying some optimization criteria for selection the of the best ensemble of classifiers from all base classifiers in the pool. Pruning involves discarding less relevant members of the pool from future combination according to different criteria. Discarding all unselected classifiers from the pool at each learning stage decreases the system performance as described in Section 3.2.2.

3.2.1. Model selection

Ensemble selection techniques are employed to choose a compact ensemble from a large pool of classifiers to increase accuracy and reduce the computational and storage costs of systems deployed during operation. A brute-force search for the optimal ensemble of classifiers results in a combinatorial complexity explosion: the search space comprises 2^K possible ensembles for a pool of K classifiers. Therefore, ensemble selection attempts to select the best ensemble of classifiers from the pool based on different optimization criteria and selection strategies [21,54]. Typical optimization criteria are ensemble accuracy [54], cross-entropy [55], and ROC-based measures [56]. Although there is no universal consensus about the definition and efficiency of diversity measures [15], certain approaches have shown promising results [56–59]. The selection strategies include ranking-based techniques in which the pool members are ordered according to an evaluation measure on a validation set, and the top classifiers are then selected to form an ensemble [60]. Search-based ensemble selection is an alternative approach which consists in combining the outputs of classifiers and then selecting the best performing ensemble evaluated on an independent validation set. A heuristic search in the space of all possible ensembles is typically performed, while evaluating the collective merit of selected members [40,55,57,58]. Typically, the selection procedure stops when a user-defined maximum number of classifiers is reached [55] or when remaining classifiers provide no further improvement to the ensemble [40,54].

Many ensemble selection approaches have been considered for cost-insensitive applications with sufficient amount of training data. Few approaches consider cost-sensitive and limited training data [56,61], which are prevalent in anomaly detection applications. Although the performance measures employed in these cost-sensitive approaches consider the cost of errors and class imbalance, they are either computed at a specific decision threshold or averaged over all decision thresholds. Any change in the operating conditions, such as prior probabilities, cost of errors, and tolerable *fpr* values, requires reconsidering the combination and selection process.

The proposed ensemble selection algorithms (BC_{greedy} and BC_{search}), described below, employ both rank- and search-based selection strategies to optimize the area under the ROCCH (AUCH). Before applying their selection strategies, they rank all available pool members in decreasing order of AUCH performance on a validation set. In contrast with existing techniques, the proposed selection algorithms exploit the monotonicity of the *incrBC* algorithm for an early stopping criterion. As show in lines 15 and 28 of Algorithm 2, *incrBC* is bound below by the previous ROCCH, and hence guarantees a monotonic improvement in AUCH performance. Furthermore, both algorithms allow for adaptation to changes in prior probabilities and costs of errors by simply shifting the desired operational point, which activates different HMMs, decision thresholds and Boolean functions.

Algorithm 3. $BC_{greedy}(\mathcal{P},\mathcal{V})$: Boolean combination with a greedy selection.

input: Pool of HMMs $\mathcal{P}=\{\lambda_1,\lambda_2,\ldots,\lambda_K\}$ and a validation set $\mathcal{V}$
output: EoHMMs (E) from the pool $\mathcal{P}$
1 **set** *tol*; // tolerated improvement in AUCH values
2 $j \leftarrow 1$
3 **foreach** $\lambda_k \in \mathcal{P}$ **do**
4 ⌊ **compute** ROC curves and their $ROCCH_k$, using $\mathcal{V}$
5 **sort** HMMs $(\lambda_1,\ldots,\lambda_K)$ in descending order of their $AUCH(ROCCH_k)$ values
6 $\lambda_j = \text{argmax}_k\{AUCH(ROCCH_k) : \lambda_k \in \mathcal{P}\}$
7 $E \leftarrow \lambda_j$; //select HMM with the largest AUCH value
8 $\mathcal{S}_j \leftarrow \lambda_j$;
9 **for** $k \leftarrow 2$ **to** K **do**
10 | $ROCCH_k = incrBC((\mathcal{S}_j,\lambda_k),\mathcal{V})$
11 | **if** $AUC(ROCCH_k) \geq AUC(ROCCH_j)+tol$ **then**
12 | | $j \leftarrow j+1$
13 | | $E \leftarrow E \cup \lambda_k$; // select the k^{th} HMM
14 | | $ROCCH_j \leftarrow ROCCH_k$; // update the convex hull
15 | | **if** $k=2$ **then**
16 | | | $\mathcal{S}_j \leftarrow \{(\lambda_j,t_i),(\lambda_k,t_{i'}),bf\}$
17 | | | // set of selected decision thresholds from each HMM and Boolean functions; derived
18 | | | from incrBC for emerging vertices
| | | **else**
| | | | $\mathcal{S}_j \leftarrow \{\mathcal{S}_{j-1}(i),(\lambda_k,t_{i'}),bf\}$
| | | | // set of selected subset from previous combinations, decision thresholds from the
| | | | newlyselected HMM, and Boolean functions; derived from incrBC for emerging vertices
19 **store** $\mathcal{S}_j$, $2 \leq j \leq K$
20 **return** E

As described in Algorithm 3, the BC_{greedy} algorithm employs a greedy search within the pool of HMMs. First, the HMMs in the pool are ranked in decreasing order of their AUCH performance on a validation set and the best HMM is selected. Then, the algorithm starts a cumulative combination of successive HMMs one at the time, selecting only those that improve the AUCH performance, of a cumulative EoHMMs, over a user-defined tolerance value. The algorithms stops when the last HMM in the pool is reached. The worst-case time complexity of this algorithm is $\mathcal{O}(K \log K)$ for sorting, followed by $\mathcal{O}(K)$ for scanning down the ensemble, resulting in $\mathcal{O}(K \log K)$ time complexity w.r.t. the number of Boolean operations of *incrBC* (see Section 3.1).

As described in Algorithm 4, the BC_{search} algorithm employs an incremental selective search strategy. It also ranks all HMMs in decreasing order of their AUCH performance on a validation set and selects the HMM with the largest AUCH value. Then, it combines the selected HMM with each of the remaining HMMs in the pool. The HMM that most improves the AUCH performance of the EoHMMs, is then selected. The cumulative EoHMMs are then combined with each of the remaining HMMs in the pool, and the HMM that provides the largest AUCH improvement to the EoHMMs is selected, and so on. The algorithm stops when the AUCH improvement of the remaining HMMs is lower than a user-defined tolerance value, or when all HMMs in the original ensemble are selected. The worst-case time complexity of this selection algorithm is $O(K^2)$ w.r.t. the number of Boolean operations of *incrBC* (see Section 3.1). However, this is only attained for a zero tolerance value for which the algorithm selects all models with the optimum order for combination. In practice, however, depending on the value of the tolerance, the selected number of models $k < K$ and also the computational time, can be traded-off as desired.

Algorithm 4. $BC_{search}(\mathcal{P},\mathcal{V})$: Boolean combination with an incremental selective search.

input: Pool of HMMs $\mathcal{P}=\{\lambda_1,\lambda_2,\ldots,\lambda_K\}$ and a validation set $\mathcal{V}$
output: EoHMMs (E) from the pool $\mathcal{P}$
1 **set** *tol*; // tolerated improvement in AUCH values
2 **foreach** $\lambda_k \in \mathcal{P}$ **do**
3 ⌊ **compute** ROC curves and their $ROCCH_k$, using $\mathcal{V}$
4 **sort** HMMs $(\lambda_1,\ldots,\lambda_K)$ in descending order of their $AUCH(ROCCH_k)$ values
5 $\lambda_1 = \text{argmax}_k\{AUCH(ROCCH_k) : \lambda_k \in \mathcal{P}\}$
6 $E \leftarrow \lambda_1$; //select HMM with the largest AUCH value

foreach $\lambda_k \in \mathcal{P}\backslash E$ **do** //remaining HMMs in $\mathcal{P}$
  $\lfloor ROCCH_k = incrBC\ ((\lambda_1,\lambda_k),\mathcal{V})$
$\lambda_2 = \text{argmax}_k\{AUCH(ROCCH_k) : \lambda_k \in \mathcal{P}\backslash E\}$
$E \leftarrow E \cup \lambda_2$; //select HMM that provides the largest AUCH improvement to E
$ROCCH_1 \leftarrow ROCCH_2$; // update the convex hull
$\mathcal{S}_2 \leftarrow \{(\lambda_1,t_i),(\lambda_2,t_{i'}),bf\}$; // Set of selected decision thresholds from each HMM and Boolean functions; derived from incrBC for emerging vertices
$j \leftarrow 3$
repeat
  foreach $\lambda_k \in \mathcal{P}\backslash E$ **do**
    $\lfloor ROCCH_k = incrBC((\mathcal{S}_{j-1},\lambda_k),\mathcal{V})$
  $\lambda_j = \text{argmax}_k\{AUCH(ROCCH_k) : \lambda_k \in \mathcal{P}\backslash E\}$
  $E \leftarrow E \cup \lambda_j$
  $ROCCH_{j-1} \leftarrow ROCCH_j$
  $\mathcal{S}_j \leftarrow \{\mathcal{S}_{j-1}(i),(\lambda_j,t_{i'}),bf\}$ // Set of selected subset from previous combinations, decision thresholds from the newlyselected HMM, and Boolean functions; derived from incrBC for emerging vertices
  $j \leftarrow j+1$
$AUCH(ROCCH_j) \le AUCH(ROCCH_{j-1}) + tol$; // no further improvement
store $\mathcal{S}_j, 2 \le j \le K$
return E

For both ensemble selection algorithms the AUCH improvement is the only user-defined parameters. If desired, it is also possible to impose a maximum number of HMMs as a stopping criterion. The performance measure employed to guide the search is not restricted to AUCH. Other measures such as the partial AUCH or the true positive rate at a required false positive rate, can be also employed for a region-specific performance. Although these performance measures summarize the ROC space with a single value to guide the search for potential ensemble members, the final composite ROCCH is always stored for visualization of the whole range of performance and adaptation of the operating point to environmental changes.

Fig. 6 presents an example illustrating the level of performance achieved by *incrBC*, BC_{greedy}, and BC_{search} algorithms. In Fig. 6, all algorithms achieve a comparable ROCCH and AUC performance. However, as shown in the legends, the size of the EoHMMs obtained by BC_{search} is four HMMs compared to seven HMMs selected by BC_{greedy} from the pool of eight HMMs, which are all combined by *incrBC*. For comparison, the selected set of decision thresholds and Boolean functions for the vertices denoted by C_1 and C_2 on the ROCCHs of Fig. 6 are shown below for each algorithms:

$$incrBC \begin{cases} C_1 = (((((((\lambda_1^1,t_1)\vee(\lambda_2^1,t_1))\wedge\neg(\lambda_3^1,t_1))\vee(\lambda_4^1,t_1))\vee(\lambda_1^2,t_2))\vee(\lambda_2^2,t_2))\wedge(\lambda_3^2,t_2))\wedge(\lambda_4^2,t_2)) \\ C_2 = (((((((\lambda_1^1,t_1)\vee(\lambda_2^1,t_1))\wedge\neg(\lambda_3^1,t_1))\vee(\lambda_4^1,t_1))\vee(\lambda_1^2,t_2))\vee(\lambda_2^2,t_2))\wedge(\lambda_3^2,t_2))\wedge(\lambda_4^2,t_2)) \end{cases}$$

$$BC_{greedy} \begin{cases} C_1 = ((((((\neg(\lambda_3^2,t_1)\wedge(\lambda_4^2,t_1))\vee(\lambda_3^1,t_1))\vee\neg(\lambda_2^2,t_1))\wedge(\lambda_2^1,t_1))\wedge\neg(\lambda_1^2,t_1))\vee(\lambda_1^1,t_1)) \\ C_2 = ((((((\neg(\lambda_3^2,t_1)\wedge(\lambda_4^2,t_1))\vee(\lambda_3^1,t_1))\vee\neg(\lambda_2^2,t_1))\wedge(\lambda_2^1,t_1))\vee(\lambda_1^2,t_2))\vee(\lambda_1^1,t_2)) \end{cases}$$

$$BC_{search} \begin{cases} C_1 = (((\neg(\lambda_3^2,t_1)\wedge(\lambda_2^2,t_1))\vee(\lambda_3^1,t_1))\vee(\lambda_1^2,t_1)) \\ C_2 = (((\neg(\lambda_3^2,t_1)\wedge(\lambda_2^2,t_1))\vee(\lambda_3^1,t_2))\vee(\lambda_1^2,t_2)) \end{cases}$$

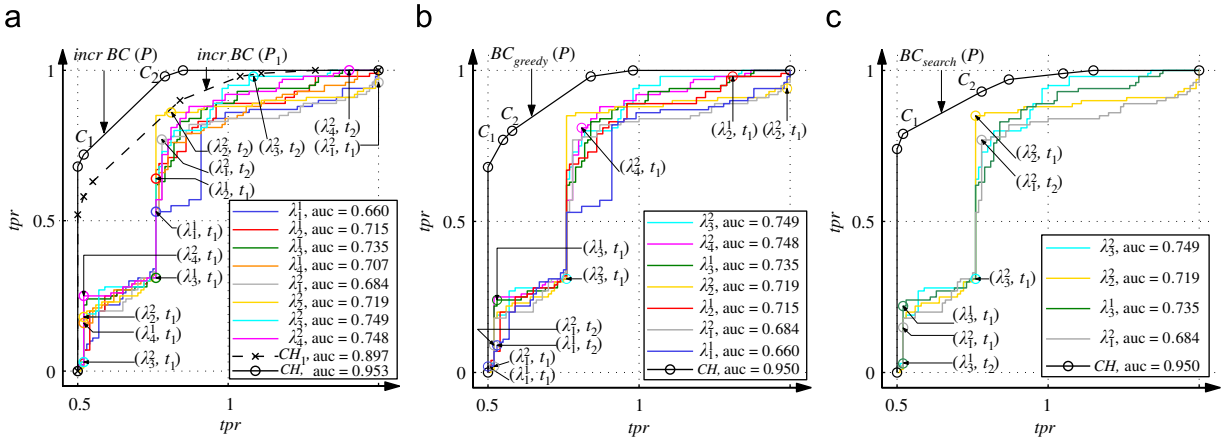


Fig. 6. A comparison of the composite ROCCH (CH) and AUC performance achieved with the HMMs selected according to the BC_{greedy} and BC_{search} algorithms (shown in the legends). Suppose that a new pool of four HMMs $\mathcal{P}_2 = \{\lambda_1^2,\ldots,\lambda_4^2\}$ is generated from a new block of training data (D_2), and appended to the previously generated pool, $\mathcal{P}_1 = \{\lambda_1^1,\ldots,\lambda_4^1\}$, in the example presented in Section 3.1 (Fig. 4). The *incrBC* algorithm combines all available HMMs in the pool according to their order of generation and storage, $\mathcal{P} = \{\lambda_1^1,\ldots,\lambda_4^1,\lambda_1^2,\ldots,\lambda_4^2\}$, while BC_{greedy} and BC_{search} start by ranking the members of $\mathcal{P}$ according to AUC values and then apply their ensemble selection strategies. (a) *incrBC*, (b) BC_{greedy} and (c) BC_{search}.

This example demonstrates the efficiency of the proposed ensemble selection techniques in forming compact EoHMMs by exploiting the new information provided from newly acquired data while maintaining a high level of performance. More extensive simulations and detailed discussions are presented in Section 5.

3.2.2. Model pruning

Pruning less relevant models is essential to restrict the pool size with incremental learning from growing indefinitely as new blocks of data become available. As described in the previous subsection, BC_{greedy} and BC_{search} algorithms are designed to form the most compact EoHMMs from a pool $\mathcal{P}$ of HMMs. According to the learn-and-combine approach new pools of HMMs are accumulated from successive blocks of data. When the number of data blocks increases over time, an increasingly large storage space is required for storing these HMMs. In addition, the time complexity of BC_{greedy} and BC_{search} algorithms also increases over time. Discarding unselected classifiers immediately from the pool yields to knowledge corruption and hence to a decline system performance. Although these classifiers did not provide any improvement to the cumulative EoHMMs, they may provide diverse information to the newly generated pool of HMMs, which have different view of the data, to increase system performance. One solution to this issue is to discard repeatedly unselected HMMs over time. A counter is therefore assigned to each HMM in the pool indicating the number of blocks for which an HMM was not selected as an ensemble member (see Algorithm 1). An HMM is then pruned from the pool, according to a user-defined life time (LT) expectancy value of unselected models. For instance, with an $LT=3$ all HMMs that have not been selected after receiving three blocks of data, as indicated by their counters, are discarded from the pool.

4. Experimental methodology

4.1. Data sets

The experiments are conducted on both synthetically generated data and sendmail data from the University of New Mexico (UNM) data sets.[3] The UNM data sets are commonly used for benchmarking anomaly detections based on system calls sequences [1]. In related work, intrusive sequences are usually labeled by comparing normal sequences, using the Sequence Time-Delay Embedding (STIDE) matching technique. This labeling process considers STIDE responses as the ground truth, and leads to a biased evaluation and comparison of techniques, which depends on both training data size and detector window size. To confirm the results on system calls data from real processes, the same labeling strategy is used in this work. However fewer sequences are used to train the HMMs to alleviate the bias. Therefore, STIDE is first trained on all the available normal data, and then used to label the corresponding sub-sequences from the ten sequences available for testing. The resulting labeled sub-sequences are concatenated, then divided into blocks of equal sizes, one for validation and the other for testing. During the experiments, smaller blocks of normal data (100–1000 sub-sequences) are used for training the HMMs as normal system call observations are very redundant. In spite of labeling issues, redundant training data, and unrepresentative test data, UNM sendmail data set is the mostly used in literature due to limited publicly available system call data sets.

The need to overcome issues encountered when using real-world data for anomaly based HIDS (incomplete data for training and labeling) has lead to the implementation of a synthetic data generation platform for proof-of-concept simulations. It is intended to provide normal data for training and labeled data (normal and anomalous) for testing. This is done by simulating different processes with various complexities then injecting anomalies in known locations. The data generator is based on the Conditional Relative Entropy (CRE) of a source; it is closely related to the work of Tan and Maxion [28]. The CRE is defined as the conditional entropy divided by the maximum entropy ($MaxEnt$) of that source, which gives an irregularity index to the generated data. For two random variables x and y the CRE is given by $CRE=-\sum_x p(x)\sum_y p(y|x)\log p(y|x)/MaxEnt$, where for an alphabet of size Σ symbols, $MaxEnt=-\Sigma\log(1/\Sigma)$ is the entropy of a theoretical source in which all symbols are equiprobale. It normalizes the conditional entropy values between $CRE=0$ (perfect regularity) and $CRE=1$ (complete irregularity or random). In a sequence of system calls, the conditional probability, $p(y|x)$, represents the probability of the next system call given the current one. It can be represented as the columns and rows (respectively) of a Markov Model with the transition matrix $M=\{a_{ij}\}$, where $a_{ij}=p(S_{t+1}=j|S_t=i)$ is the transition probability from state i at time t to state j at time $t+1$. Accordingly, for a specific alphabet size Σ and CRE value, a Markov chain is first constructed, then used as a generative model for normal data. This Markov chain is also used for labeling injected anomalies as described below. Let an anomalous event be defined as a surprising event which does not belong to the process normal pattern. This type of event may be a *foreign-symbol* anomaly sequence that contains symbols not included in the process normal alphabet, a *foreign n-gram* anomaly sequence that contains *n-grams* not present in the process normal data, or a *rare n-gram* anomaly sequence that contains *n-grams* that are infrequent in the process normal data and occurs in burst during the test.[4]

Generating training data consists of constructing Markov transition matrices for an alphabet of size Σ symbols with the desired irregularity index (CRE) for the normal sequences. The normal data sequence with the desired length is then produced with the Markov chain, and segmented using a sliding window (shift one) of a fixed size, DW. To produce the anomalous data, a random sequence ($CRE=1$) is generated, using the same alphabet size Σ, and segmented into sub-sequences of a desired length using a sliding window with a fixed-size of AS. Then, the original generative Markov chain is used to compute the likelihood of each sub-sequence. If the likelihood is lower than a threshold it is labeled as anomaly. The threshold is set to $(\min(a_{ij}))^{AS-1},\forall_{i,j}$, the minimal value in the Markov transition matrix to the power ($AS-1$), which is the number of symbol transitions in the sequence of size AS. This ensures that the anomalous sequences of size AS are not associated with the process normal behavior, and hence foreign *n-gram* anomalies are collected. The trivial case of foreign-symbol anomaly is disregarded since it is easy to be detected. Rare *n-gram* anomalies are not considered since we seek to investigate the performance at the detection level, and such kind of anomalies are accounted for at a higher level by

[3] http://www.cs.unm.edu/~immsec/systemcalls.htm.

[4] This is in contrast with other work which consider rare event as anomalies. Rare events are normal, however they may be suspicious if they occur in high frequency over a short period of time.

computing the frequency of rare events over a local region. Finally, to create the testing data another normal sequence is generated, segmented and labeled as normal. The collected anomalies of the same length are then injected into this sequence at random according to a mixing ratio.

4.2. Experimental protocol

The experiments conducted in this paper using the data generator simulate a simple process, with $\Sigma = 8$ and $CRE = 0.3$ and a more complex process, with $\Sigma = 50$ and $CRE = 0.4$. The sizes of injected anomalies are assumed equal to the detector window sizes $AS = DW = 4$. For both scenarios, the presented results are for validation and test sets that comprise 75% of normal and 25% of anomalous data. Although not show in this paper, various experiments have been conducted using different values of *AS* and *DW*, and different ratios of normal to anomalous data. These experiments produced similar results and hence the discussion presented in the next section hold.

Fig. 7 illustrates the steps involved for estimating HMM parameters. Given the first block of training data D_1, different discrete-time ergodic HMMs are trained with various number of hidden states $N = [N_{min}, \ldots, N_{max}]$ using the 10-fold cross validation (10-FCV). The training block D_1, which only comprises normal sub-sequences, is randomly partitioned into $K = 10$ sub-blocks of equal size. For each fold k, each of the learning techniques described below is used to estimate HMM parameters using $K-1$ sub-blocks. The Forward algorithm is then used to evaluate the log-likelihood on the remaining sub-block, which is used as a stopping criterion to reduce the overfitting effects. The training process is repeated ten times using a different random initialization to avoid local maxima, which provides 100 ROC curves for each N value. Finally, the model that gives the highest area under its convex hull on a validation set ($\mathcal{V}$) comprising normal and anomalous sub-sequences is selected, which results an HMM for each N value.

Fig. 8 presents HMM parameter estimation according to each learning technique from successive blocks of data for each N value. When the second training block D_2 becomes available, the batch Baum–Welch (BBW) algorithm discards the previously learned HMMs and restarts the training procedure with all cumulative data ($D_1 \cup D_2$), while the Baum–Welch (BW) algorithm restarts the training using D_2 only providing HMMs for incremental combination according to the *incrBC* algorithm. The on-line BW (OBW) and incremental BW (IBW) algorithms resume the training from the previously learned HMMs (λ_N^1) using only the current block (D_2). The OBW algorithm re-estimates HMM parameters based on each sub-sequence without iterations, while the IBW algorithm re-iterates on D_2 until the stopping criteria are met [10].

Assume that each block of training data D_t (acquired at a time interval t) comprises R sub-sequences of observation symbols, each of length *DW*. During the design phase, the worst-case time complexity of an HMM trained according to BBW on cumulative blocks of data ($D_1 \cup D_2, \ldots \cup D_t$) is $\mathcal{O}(I.N^2.t.R.DW)$, where I is the maximum number of iterations allowed. An HMM trained according to the remaining algorithms (BW, IBW, or OBW) uses only D_t. The worst-case time complexity of an HMM trained according to *BW* or *IBW* algorithm is

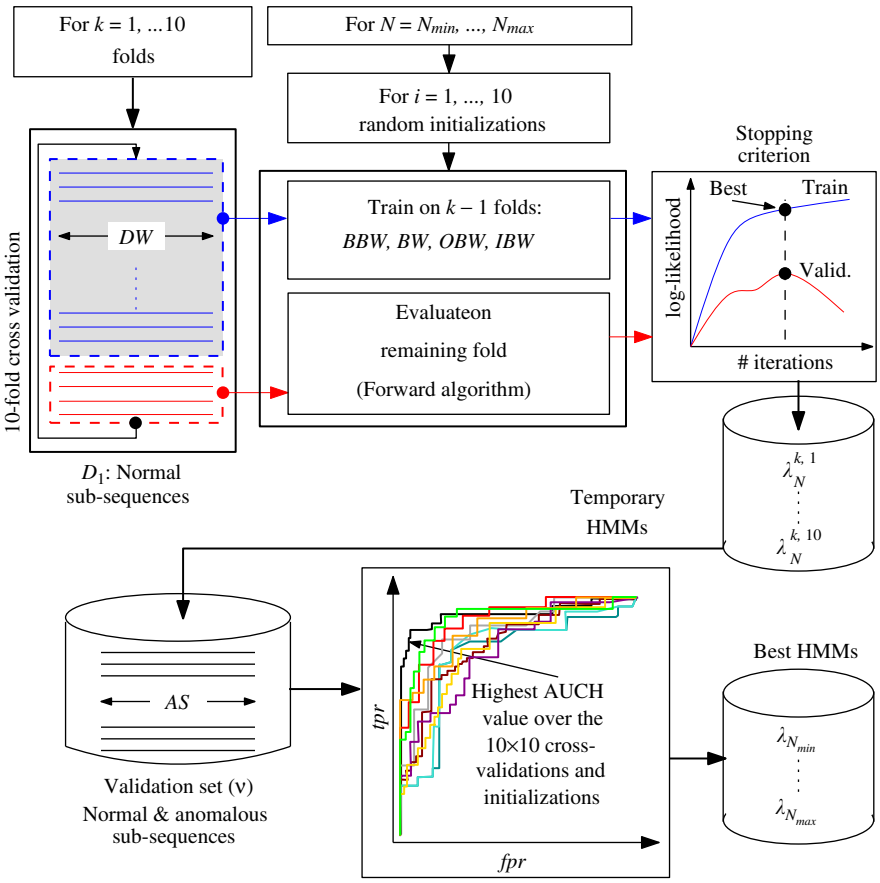


Fig. 7. Overall steps involved to estimate HMM parameters and select HMMs with the highest AUCH for each number of states from the first block (D_1) of normal data, using 10-FCV and ten random initializations.

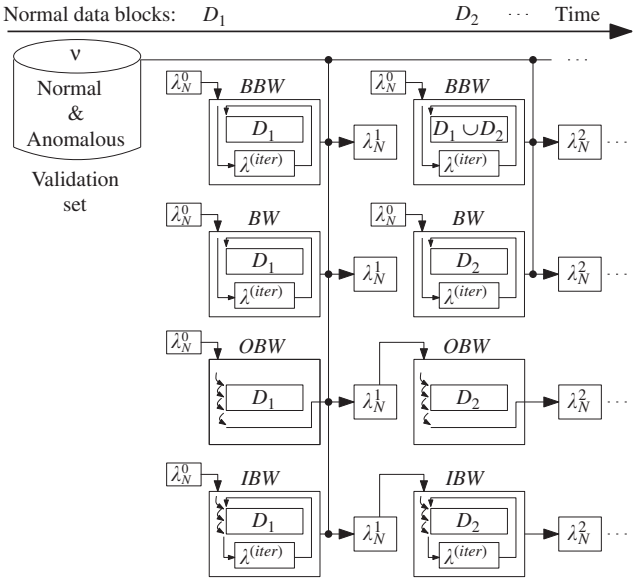


Fig. 8. An illustration of HMM parameter estimation according to each learning technique (BBW, BW, OBW, and IBW) when subsequent blocks of observation sub-sequences (D_1,D_2,D_3,...) become available.

$\mathcal{O}(I.N^2.R.DW)$, and that of *OBW* is $\mathcal{O}(N^2.R.DW)$. Therefore, *OBW* achieves the fastest training since it does not iterate over the sub-sequences in the training block D_t, followed by *IBW* and *BW*.

When the first block is provided for training, *OBW* and *IBW* require training K HMMs in order to select the "best" number of state. When provided with a new block of data, those algorithms update the parameters of the (single) previously trained HMM with a fixed number of state N. On the other hand for each new block D_t, K HMMs with different number of states and initializations are trained according to *BBW* (to select the best model for operations), and also according to *BW* (to combine their response according the learn-and-merge approach). When t becomes large over time, *BBW* training time becomes prohibitively costly (using $D_1 \cup D_2, \ldots \cup D_t$), while the computational time of *BW* training remains constant (using D_t only). The additional time complexity required for incremental Boolean combination of HMMs depends on the number of accumulated HMMs in the pool ($|\mathcal{P}_t| = t.K$) and the size of the validation set $\mathcal{V}$, as described in Section 3.1.

However, operating an EoHMMs with an increasing size ($|E_t| = |\mathcal{P}_t|$) will significantly decrease the system efficiency over time, due to the time required for these HMMs to evaluate the likelihood values of the test sub-sequence. The proposed selection algorithms (BC_{search} and BC_{greedy}) provide efficient solutions (see Section 3.2) that significantly decrease the number of selected HMMs, providing a small (or fixed) size EoHMMs for operations ($|E_t| \ll |\mathcal{P}_t|$), without negatively affecting system accuracy. Furthermore, the pruning strategy eliminates less accurate HMMs from the pool ($|\mathcal{P}_t| \ll |\mathcal{P}_{t-1}|$), reducing thereby the storage requirements for HMM parameters.

The area under the ROC curve (AUC) has been largely suggested as a robust scalar summary of classifiers performance [24,62]. The AUC assesses ranking in terms of class separation—it evaluates how well a classifier is able to sort its predictions according to the confidence they are assigned. For instance, with an $AUC=1$ all positives are ranked higher than negatives indicating a perfect discrimination between classes. A random classifier has an $AUC=0.5$ that is both classes are ranked at random. If the AUC or the partial AUC [63] are not significantly different, the shape of the curves might need to be looked at. It may also be useful to observe the *tpr* for a fixed *fpr* of particular interest. Since the MRROC can be applied to any ROC curve, the performance in all experiments are measured with reference to the ROCCH, including the area under the convex hull (AUCH) and the *tpr* at $fpr=0.1$. When working with the synthetic data, the whole procedure is replicated ten times with different training, validation and testing sets, and the median results are presented along with the lower and upper quartiles to provide statistical confidence intervals.

5. Simulation results

The first subsection presents the performance of the proposed learn-and-combine approach for incremental learning of new data without employing a model management strategy. In this case, a pool of HMMs for a new block of training data is combined with all previously generated HMMs according to the *incrBC* algorithm. The second subsection shows the impact on performance of the ensemble selection algorithms and of the pruning strategies for limiting the pool size.

5.1. Evaluation of the learn-and-combine approach

5.1.1. HMMs trained with a fixed number of states on successive blocks of data

The first experiment involves ergodic HMMs trained with $N=6$ states on ten blocks of data. The training, validation and testing data are generated synthetically as described in Section 4.1, with $\Sigma = 8$ and $CRE=0.3$. Each training block D_k ($k=1,\ldots,10$) comprises 50 normal sub-sequences, each of size $DW=4$. Each validation ($\mathcal{V}$) and test ($\mathcal{T}$) set comprises 200 sub-sequences, each of size $AS=4$. In both data sets, the ratio of normal to anomalous sub-sequences is 4:1. The training and validation of HMMs follow the methodology described in Section 4.2. For each block D_k, a pool $\mathcal{P}_k$ is obtained by applying the BW algorithm to D_k using 10-FCV and ten different random initializations, and selecting the HMM ($\lambda^k_{N=6}$) that gives the highest AUCH on $\mathcal{V}$ (see Fig. 7). $\mathcal{P}_k$ is then appended to a pool $\mathcal{P}$ ($\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_k$).

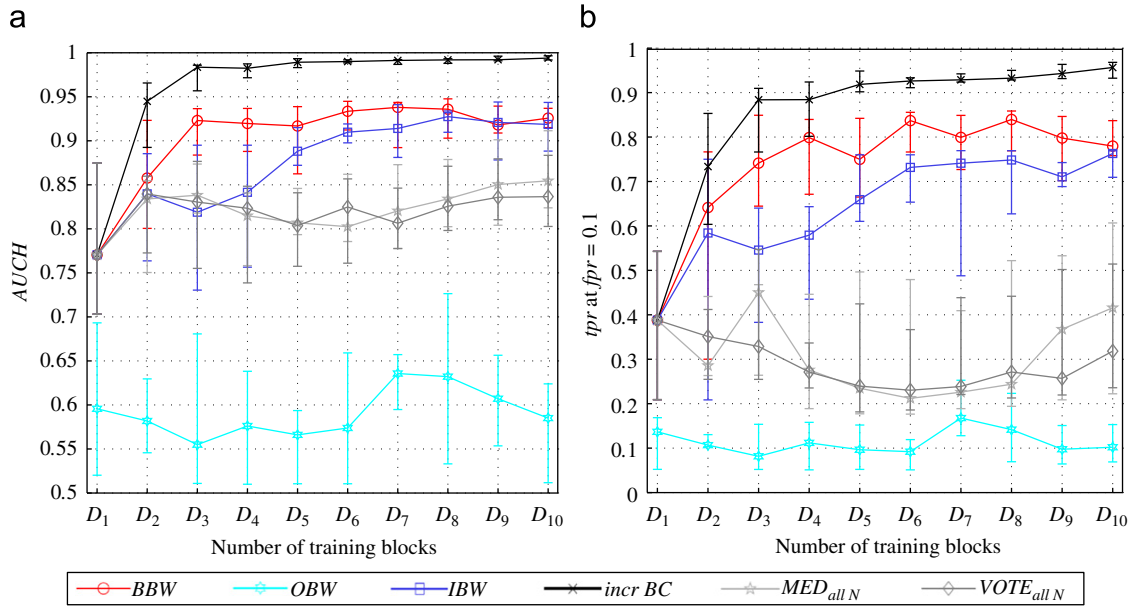


Fig. 9. Results for synthetically generated data with $\Sigma=8$ and $CRE=0.3$. The HMMs are trained according to each technique with $N=6$ states for each block of data providing a pool of size $|\mathcal{P}|=1,2,\ldots,10$ HMMs. Error bars are lower and upper quartiles over ten replications.

After receiving the last block of data, the pool $\mathcal{P}=\{\lambda^1_{N=6},\ldots,\lambda^{10}_{N=6}\}$ of size $|\mathcal{P}|=10$ HMMs is provided for incremental combination according to the *incrBC* algorithm. The reference BBW follows the same training procedure but with cumulative blocks of data, and both OBW and IBW algorithms resume the training from the previously learned HMMs using only the current block of data (see Fig. 8). Fig. 9 compares the median AUCH performance (Fig. 9(a)) and the median *tpr* values at $fpr=0.1$ (Fig. 9(b)) as well as their lower and upper quartiles over ten replications for each learning technique. The performance obtained with BBW, OBW and IBW algorithms are compared to that of the *incrBC* algorithm combining the responses of the HMMs in $\mathcal{P}$, incrementally, over the ten blocks of data. The performance achieved by combining the outputs of the HMMs in $\mathcal{P}$, over the ten blocks of data, with the static median (MED) and majority vote (VOTE) fusion functions are also provided for reference.

As shown in Fig. 9, the learn-and-combine approach using the *incrBC* algorithm provides the highest level of performance (with the lowest variances) among all incremental learning techniques over the whole range of results. Performance is significantly higher than that of the reference BBW, especially when provided with limited training data, as shown in the performance achieved with the first few blocks. The level of performance provided by the static MED and VOTE combiners is lower (with higher variances) versus other techniques, and oscillates around their initial values. Not surprisingly, the OBW algorithm has achieved the worst level of performance as one pass over a limited data is insufficient to capture its underlying structure. In contrast, the IBW algorithm has provided a higher level of performance than that of OBW as it iterates over each block and employs a fixed learning rate to integrate the newly acquired information into HMM parameters [10].

5.1.2. *HMMs trained with different number of states on successive blocks of data*

In order to investigate the effects of the number of states on the performance of each technique, the previous experiment is conducted with a number of states ranging from $N=4$ to 12. For each N value, a pool $\mathcal{P}_N$ is obtained by applying the BW algorithm to each block D_k using 10-FCV and ten different random initializations, and selecting the HMM (λ^k_N) that gives the highest AUCH on $\mathcal{V}$ (see Figs. 7 and 8). After receiving the last block of data, nine pools are generated, a pool $\mathcal{P}_N=\{\lambda^1_N,\ldots,\lambda^{10}_N\}$ for each state value $N=4,\ldots,12$, each of size $|\mathcal{P}_N|=10$ HMMs. The responses of HMMs in each pool $\mathcal{P}_N$ are incrementally combined using the *incrBC* algorithm over each block. The number of states that achieved the highest average level of performance on each block of data is selected. The same procedure is also applied for BBW, OBW and IBW algorithms (see Figs. 7 and 8). In addition, the learn-and-combine approach is employed to incrementally combine the HMMs trained using the whole range of states over the ten blocks. The nine pools are therefore concatenated into one pool $\mathcal{P}=\{\lambda^1_{N=4},\ldots\lambda^1_{N=12};\ldots;\lambda^{10}_{N=4},\ldots\lambda^{10}_{N=12}\}$ of size $|\mathcal{P}|=90$ HMMs, and incrementally combined according to the *incrBC* algorithm over all the successive blocks ($incrBC_{all\ N}$). The HMMs in $\mathcal{P}$ are also combined according to the median ($MED_{all\ N}$) and majority vote ($VOTE_{all\ N}$) functions for comparison. Fig. 10 presents the median results of the experiments as well as their lower and upper quartiles over ten replications for each learning technique. For BBW, OBW, IBW, and *incrBC* technique, the number of states that achieved the highest average level of performance on each block of data is indicated with each median value.

As shown in Fig. 10, the best number of states varies from one block to the next and different among all techniques. Therefore, combination of HMMs each trained with different number of states and random initializations may increase the ensemble diversity and improve system performance. This is clearly seen in the performance achieved with $incrBC_{all\ N}$, which is significantly higher than all other techniques. The previous observations hold true for the remaining techniques. In fact, to obtain the number of states which provides the best performance for each learning techniques on each data block, the HMMs are trained and evaluated with each N values according to different random initializations. Instead of selecting the single "best" HMM from the pool, $incrBC_{all\ N}$ combines all HMMs with a minimal overhead. As expected, the level of performance achieved by the learning algorithms increases when provided with more training blocks. Although *incrBC* and $incrBC_{all\ N}$ algorithms are still capable of achieving the highest level of performance, the increased performance over other techniques is relatively lower, as the combined HMMs become more positively correlated.

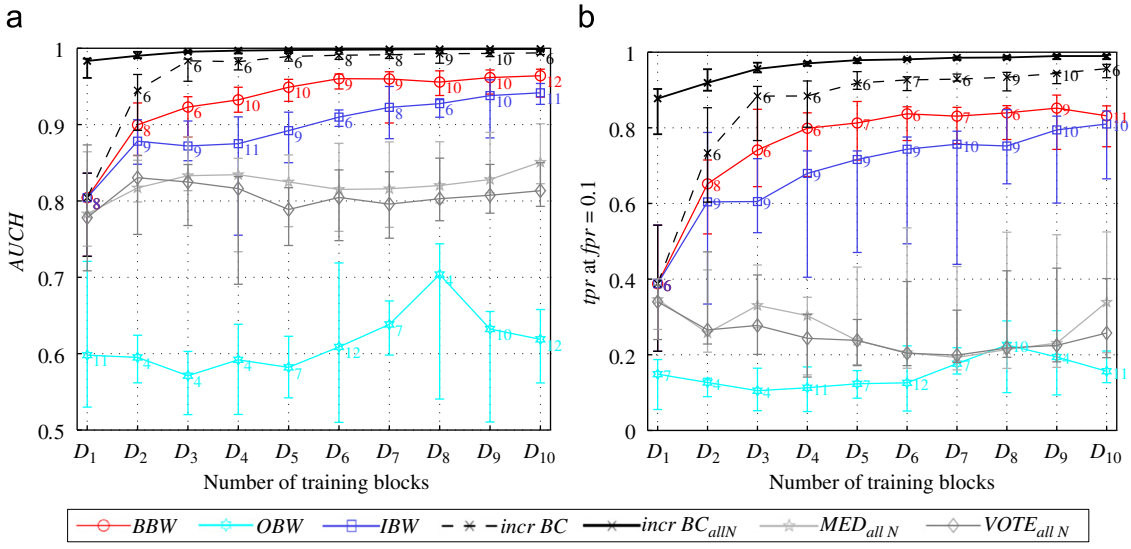


Fig. 10. Results for synthetically generated data with $\Sigma = 8$ and $CRE = 0.3$. The HMMs are trained according to each technique with nine different states ($N = 4,5,\ldots,12$) for each block of data providing a pool of size $|\mathcal{P}| = 9,18,\ldots,90$ HMMs. Numbers above points are the state values that achieved the highest average level of performance on each block. Error bars are lower and upper quartiles over ten replications.

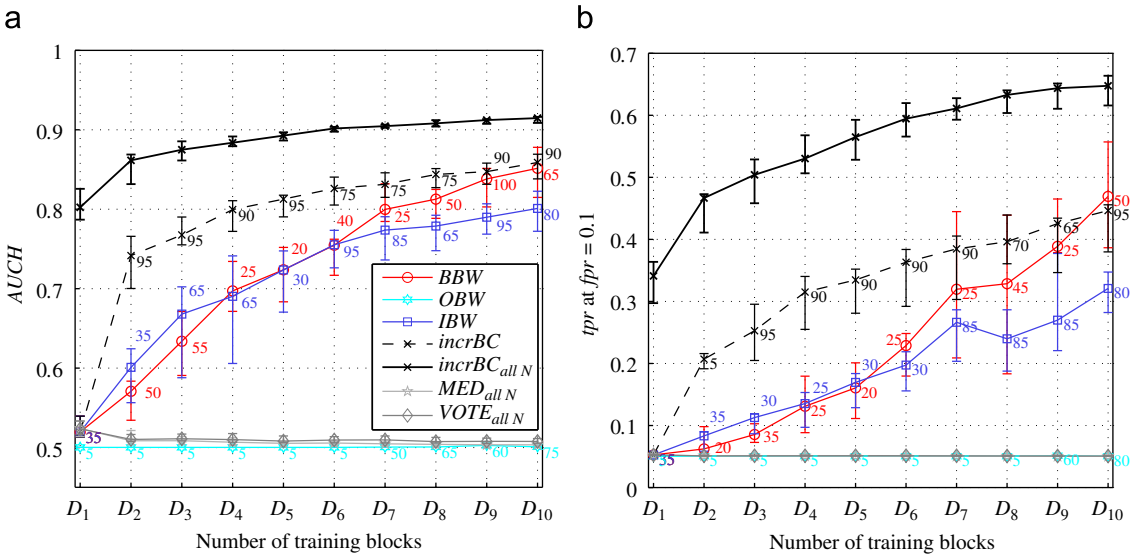


Fig. 11. Results for synthetically generated data with $\Sigma = 50$ and $CRE = 0.4$. The HMMs are trained according to each technique with 20 different states ($N = 5,10,\ldots,100$) for each block of data providing a pool of size $|\mathcal{P}| = 20,40,\ldots,200$ HMMs. Numbers above points are the state values that achieved the highest average level of performance on each block. Error bars are lower and upper quartiles over ten replications.

The previous results are also confirmed on the more complex synthetic data in Fig. 11 and on sendmail data in Fig. 12. The training and validation of the ergodic HMMs with 20 different number of states ($N = 5,10,\ldots,100$) is carried out according to the methodology described in Section 4. For each of the ten data blocks, 20 HMMs are generated and appended to the pool, which gives a pool of size $|\mathcal{P}| = 20,40,\ldots,200$ HMMs. For each replication of the synthetic data, the training is conducted on ten successive blocks each comprising 500 sub-sequences of length $DW = 4$ symbols, the validation set $\mathcal{V}$ comprises 2000 sub-sequences and the test set $\mathcal{T}$ comprises 5000 sub-sequences. For sendmail data, the training is conducted on ten successive blocks each comprising 100 sub-sequences of length $DW = 4$ symbols, each $\mathcal{V}$ and $\mathcal{T}$ comprise 450 sub-sequences. In both cases, the anomaly size is $AS = 4$ symbols and the ratio of normal to anomalous sequences is 4:1. Again, the incremental learn-and-combine approach provides a significantly higher level of performance than the BBW, OBW, IBW, MED and VOTE techniques.

The level of performance achieved by applying the learn-and-combine approach to HMMs trained on each newly acquired block of data always provide the highest overall accuracy. In particular, the results have shown that it provides a higher level of performance than the reference BBW, especially when provided with limited data. This stems from the capabilities of the *incrBC* algorithm in effectively exploiting the diverse and complementary information provided from the pool of HMMs trained with different number of states and different initializations, and from the newly acquired data. In fact, BW training optimizes HMM parameters locally, which results in converging to different local maxima. Furthermore, HMMs trained with a different number of states allow for capturing different structures of the underlying data. In addition, the newly acquired blocks of training data provide different views of the true underlying distribution. According to the bias-variance decomposition [18,64], segmenting the training data introduces bias and decreases the generalization ability of the individual classifiers. However, this increases the variances among classifiers trained on each data block.

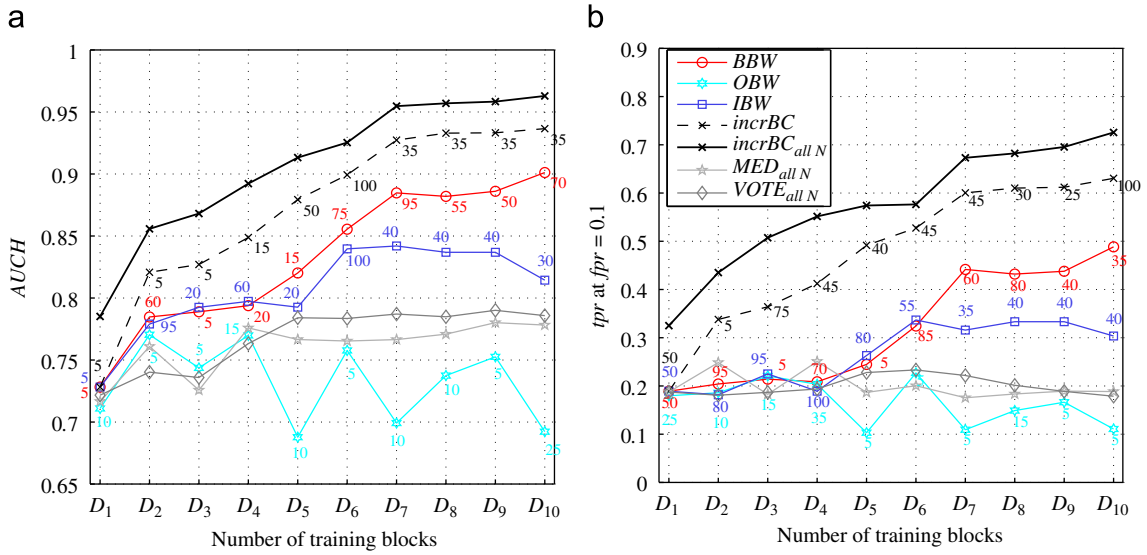


Fig. 12. Results for sendmail data. The HMMs are trained according to each technique with 20 different states ($N=5,10,\ldots,100$) for each block of data providing a pool of size $|\mathcal{P}|=20,40,\ldots,200$ HMMs. Numbers above points are the state values that achieved the highest level of performance on each block.

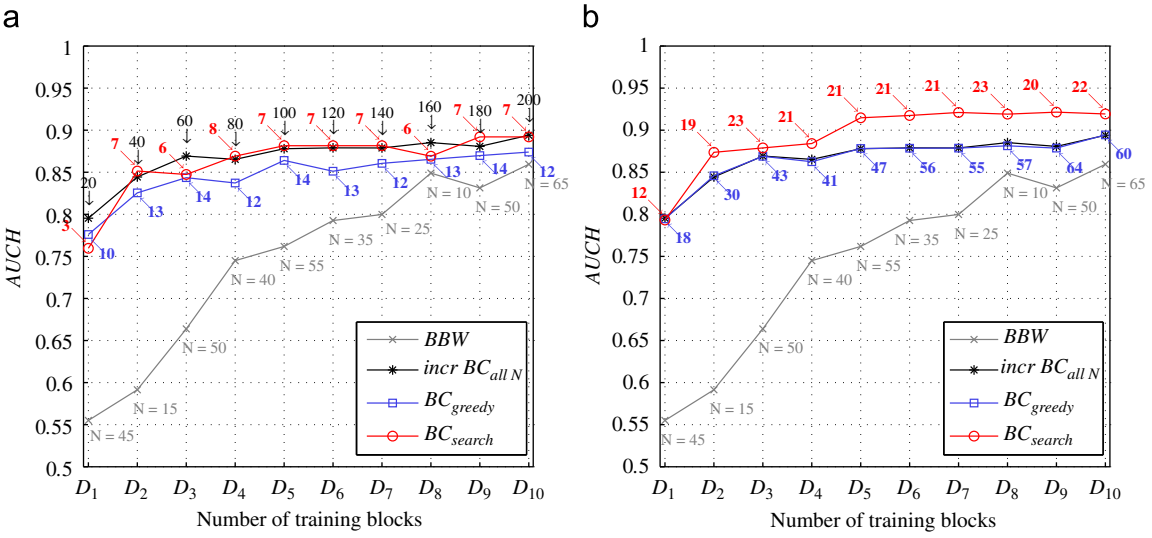


Fig. 13. Ensemble selection results for synthetically generated data with $\Sigma=50$ and $CRE=0.4$. These results are for the first replication of Fig. 11. For each block, the values on the arrows represent the size of the EoHMMs ($|E|$) selected by each technique from the pool of size $|\mathcal{P}|=20,40,\ldots,200$ HMMs.

An increased generalization ability is therefore achieved by combining the outputs of those classifiers and hence improved level of performance.

The MED and VOTE fusion functions have shown incapable of increasing the level of performance compared to the *incrBC* technique. This reflects their inabilities to effectively exploit the information provided from the validation set. The MED function directly combines HMMs likelihood values for each sub-sequence in the test, while VOTE considers the crisp decisions from HMMs at optimal operating thresholds or equivalently at equal error rates. In contrast, the *incrBC* algorithm applies ten Boolean functions to the crisp decisions provided by each threshold from the first HMM to those provided by the second HMM, and then selects the decision thresholds and Boolean functions that improve the overall ROCCH of the validation set $\mathcal{V}$. Shifting the decision thresholds for each HMM in the ensemble before combining their responses has shown to largely increase the diversity of the ensemble. The *incrBC* algorithm is effectively designed to take advantage from these threshold-based complementary information and to select those that most improve the level of performance.

5.2. Evaluation of model management strategies

5.2.1. Model selection

In the previously conducted experiments, using the synthetic ($\Sigma=50$ and $CRE=0.4$) and sendmail data sets, the pool $\mathcal{P}$ comprised 20 HMMs for each data block (an HMM for each number of states $N=5,10,\ldots,100$), yielding a pool of size $|\mathcal{P}|=200$ HMMs after receiving the ten blocks of data. This is also the size of the EoHMMs ($|E|=20,40,\ldots,200$ HMMs), since for each block D_k, all available HMMs in $\mathcal{P}$ were selected and incrementally combined according to the learn-and-combine approach ($incrBC_{all\ N}$). Fig. 13 presents the results of the combined EoHMMs previously considered using the synthetically generated data with $\Sigma=50$ and $CRE=0.4$. These results belong to the

first replication of Fig. 11. AUCH performance of the $incrBC_{all\ N}$ that combines all available HMMs from the pool is compared to that of the proposed ensemble selection algorithms, BC_{greedy} (Algorithm 3) and BC_{search} (Algorithm 4). For each block, the size of the EoHMMs selected according to each technique is shown on the respective curves of the figure. The performance of the BBW, with the selected (best) state value at each block, is also shown for reference.

Fig. 13(a), presents the results of the BC_{greedy} and BC_{search} algorithms for a tolerated improvement value of 0.01 between the AUCH values. Both BC_{greedy} and BC_{search} are capable of maintaining a slightly lower level of AUCH performance than that of the $incrBC_{all\ N}$ algorithm. However, for each block, the size of the selected EoHMMs ($|E|$) is largely reduced compared with the original pool size ($|\mathcal{P}| = 20,40,\ldots,200$ HMMs). The BC_{greedy} algorithm selects ensembles of sizes $|E| = 10$ to 14 HMMs, while BC_{search} selects ensembles of sizes from $|E| = 3$ to 8 HMMs. When the number of blocks increases, the number of the selected HMMs according to both algorithms remains almost stable. In particular, at the 10th block of data, BC_{greedy} selects an ensemble of size $|E| = 12$ HMMs, while BC_{search} selects an ensemble of size $|E| = 7$ from the generated pool of size $|\mathcal{P}| = 200$ HMMs, and even provides a slight increase in AUCH performance over that of $incrBC_{all\ N}$. This indicates that both BC_{greedy} and BC_{search} are effective in exploiting the complimentary information provided from the new blocks of data.

As expected, the incremental search strategy employed within the BC_{search} algorithm is most effective for reducing the ensemble sizes while maintaining or improving the overall performance. The selection strategy employed within the BC_{greedy} algorithm is more conservative than that of the BC_{search}. BC_{greedy} tends to select and conserve older models due to its linear scanning and selection, while the incremental search strategy of BC_{search} exploits further information by exploring the benefit achieved after combining each new HMM with the cumulative EoHMMs. In fact, the order in which the models are selected in BC_{search} assures the best ensemble performance up to the tolerance value. Therefore, as illustrated in Fig. 13(b), with lower tolerance values, the level of performance achieved by BC_{search} is higher than that of BC_{greedy} and $incrBC_{all\ N}$ algorithms. In this case, the size of the EoHMMs selected according to BC_{search} is on average about half of that selected by BC_{greedy}, over the ten blocks. However, this comes at the cost of efficiency, where BC_{search} may be an order of magnitude more computationally expensive than BC_{greedy} (see Section 3.2). These selection results are also confirmed on sendmail data as shown in Fig. 14. In practice, when efficiency is important, BC_{greedy} should be considered as it scans the ordered list of detectors once. Otherwise, BC_{search} has shown to be more effective in selecting detectors that contribute the most to an improved performance of the ensemble.

5.2.2. Model pruning

Previous experiments have shown that BC_{greedy} and BC_{search} algorithms are effective in maintaining a high level of performance by selecting relatively small sized EoHMMs from the entire pool of previously generated HMMs. In practice, however, the size of the pool must be restricted from increasing indefinitely with the number of data blocks. The impact on performance of the pruning strategy proposed in Section 3.2 is now evaluated for BC_{greedy} and BC_{search} algorithms according to various life time (LT) expectancy values. In the following results, an HMM is pruned if it is not selected for an LT corresponding to 1,3 or 5 data blocks. The reference results of previous experiments, which have been conducted without pruning HMMs from the pool, are shown with an $LT = \infty$.

Fig. 15 illustrates the impact on accuracy of pruning the pool of HMMs in Fig. 13(a) (synthetic data with $\Sigma = 50$ and $CRE = 0.4$), while Fig. 16 illustrates the impact of pruning the pool of HMMs in Fig. 14(a) (sendmail data). The size of the selected EoHMMs ($|E|$) and of the pool ($|\mathcal{P}|$) are presented below the figures for each block of data. The performance of the BBW algorithm and that of $incrBC_{all\ N}$, combining all HMMs without any pruning, is also shown for reference. As shown in Fig. 15, the level of performance achieved with BC_{greedy} and BC_{search} algorithms is decreased for $LT = 1$ compared to that achieved with $LT = \infty$. This aggressive pruning strategy eliminates all HMMs that have not been selected as ensemble members during the previous incremental learning stage, which may lead to knowledge corruption, and hence degrades the system performance. In fact, the discarded HMMs may have complementary information with respect to the newly generated HMMs from new block of data. Early elimination of this information limits the search space for future combinations. The impact on performance of pruning depends on the variability of the data and the block size. As shown in Fig. 16,

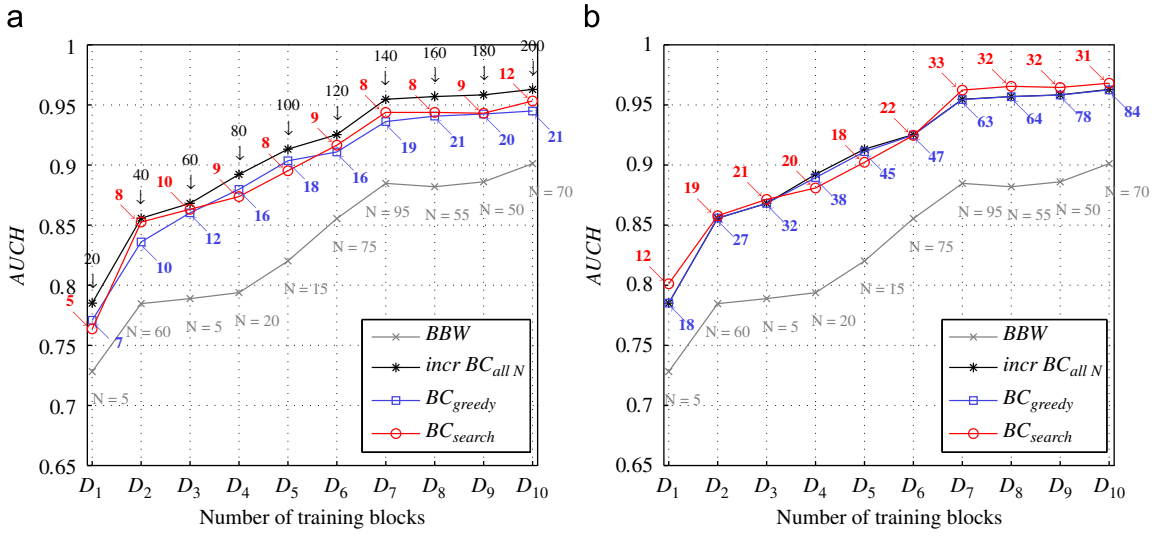
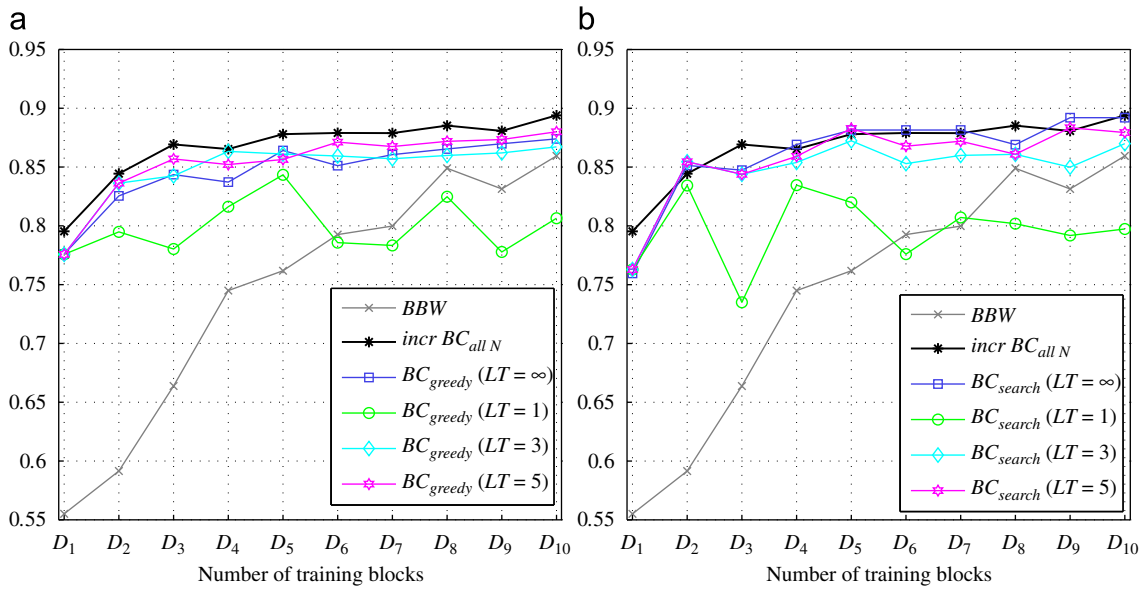


Fig. 14. Ensemble selection results for sendmail data of Fig. 12. For each block, the values on the arrows represent the size of the EoHMMs ($|E|$) selected by each technique from $\mathcal{P}$ of size $|\mathcal{P}| = 20,40,\ldots,200$ HMMs. (a) Tolerance=0.003 and (b) tolerance=0.0001.



	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
	$BC_{greedy}(LT=\infty)$									
$\lvert E\rvert$	10	13	14	12	14	13	12	13	14	12
$\lvert\mathcal{P}\rvert$	20	40	60	80	100	120	140	160	180	200
	$BC_{greedy}(LT=1)$									
$\lvert E\rvert$	7	7	7	7	10	10	7	7	8	8
$\lvert\mathcal{P}\rvert$	20	22	24	28	25	25	25	26	26	26
	$BC_{greedy}(LT=3)$									
$\lvert E\rvert$	7	7	9	9	9	9	9	9	10	10
$\lvert\mathcal{P}\rvert$	20	40	53	61	55	54	53	56	58	57
	$BC_{greedy}(LT=5)$									
$\lvert E\rvert$	7	7	9	9	10	10	8	8	13	13
$\lvert\mathcal{P}\rvert$	20	40	60	80	94	104	99	97	95	94

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
	$BC_{search}(LT=\infty)$									
$\lvert E\rvert$	3	7	6	8	7	7	7	6	7	7
$\lvert\mathcal{P}\rvert$	20	40	60	80	100	120	140	160	180	200
	$BC_{search}(LT=1)$									
$\lvert E\rvert$	6	6	7	7	3	3	7	7	3	3
$\lvert\mathcal{P}\rvert$	20	22	24	23	25	23	25	25	24	25
	$BC_{search}(LT=3)$									
$\lvert E\rvert$	6	6	7	7	7	7	3	3	7	7
$\lvert\mathcal{P}\rvert$	20	40	52	57	49	50	50	46	47	52
	$B_{search}(LT=5)$									
$\lvert E\rvert$	6	6	7	7	7	7	8	8	8	8
$\lvert\mathcal{P}\rvert$	20	40	60	80	92	97	92	93	94	92

Fig. 15. Illustration of the impact on performance of pruning the pool of HMMs in Fig. 13(a). (a) BC_{greedy} and (b) BC_{search}.

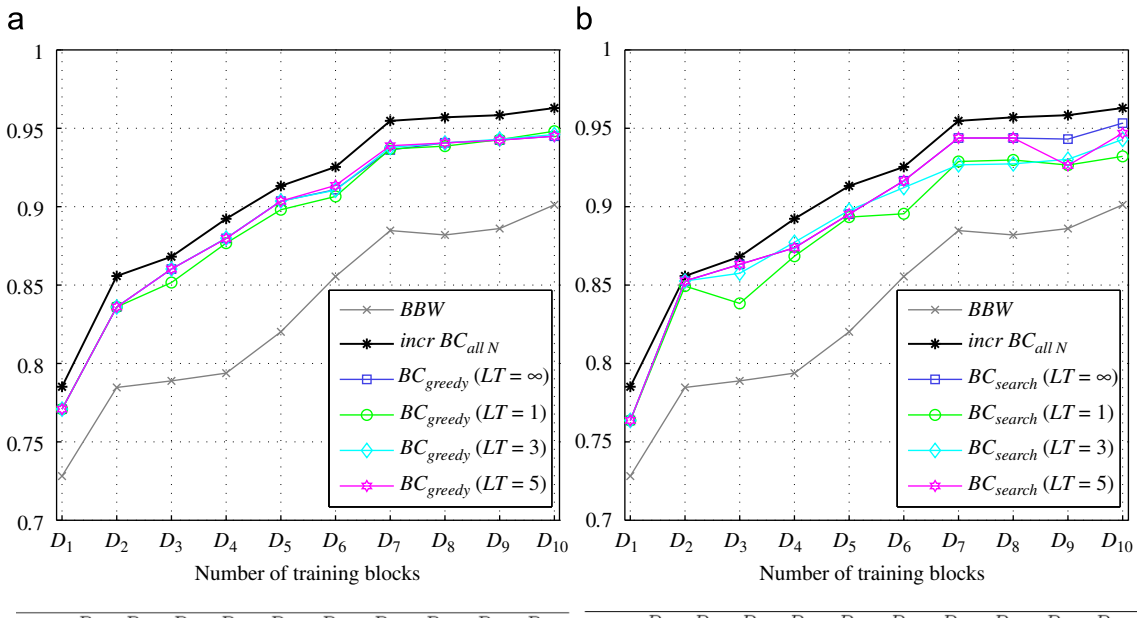
the decline in performance with $BC_{greedy}(LT=1)$ and $BC_{search}(LT=1)$ is relatively small for sendmail data, which incorporates more redundancy than the synthetically generated data.

The performance achieved with a delayed pruning of HMMs approaches that of retaining all generated HMMs in the pool for larger LT values. As shown in Figs. 15 and 16, the performance achieved by pruning the HMMs that have not been selected for $LT=3$ and 5 blocks according to BC_{greedy} and BC_{search} algorithms is comparable to that of $BC_{greedy}(LT=\infty)$ and $BC_{search}(LT=\infty)$, respectively. For fixed tolerance and LT values, BC_{search} is capable of selecting smaller EoHMMs and further reducing the size of the pool than BC_{greedy} algorithm. The results show that BC_{search} limits the size of pool to about $LT+1$ times the averaged number of generated HMMs from new blocks, while BC_{greedy} upper bound on pool size is slightly higher. For instance, with $LT=5$ we need a storage that accommodates parameters of about 100 HMMs. A fixed-size pool may be obtained by adaptively changing the tolerance and LT values upon receiving a new block of data. In HMM-based ADSs, the system administrator must set these parameters to optimize the system performance, while reducing the size of the selected EoHMMs and of the pool of HMMs for reduced time and memory requirements.

6. Conclusions

This paper presents a ROC-based system to efficiently adapt EoHMMs over time, from new training data, according to a learn-and-combine approach. When a new block of data becomes available, a pool of base HMMs is generated and combined to a global pool comprising previously generated pools. The base HMMs are trained from each new data block with different number of states and initializations, which allows to capture different underlying structures of the data and hence increases diversity among pool members. The responses from these newly trained HMMs are then combined with those of the previously trained HMMs in ROC space using a novel incremental Boolean combination (*incrBC*) technique. On its own, *incrBC* allows to maintain or improve system accuracy, yet it retains all previously generated HMMs in the pool.

Specialized model management algorithms are proposed to limit the computational and memory complexity from increasing indefinitely with the *incrBC* of new HMMs. First, the proposed system selects a diversified EoHMMs from the pool according to one of two ensemble selection algorithms, called BC_{greedy} and BC_{search} that are adapted to benefit from the monotonicity in *incrBC* accuracy, for a reduced complexity. Decision thresholds from selected HMMs and Boolean fusion functions are then adapted to improve overall system performance. Finally, redundant and inaccurate base HMMs, which have not been selected for some user-defined time interval, are pruned from the pool. Since the overall composite ROC convex hull is retained, the proposed system is capable of changing its desired operating point during operations, and hence accounts for changes in operating conditions, such as tolerated false alarm rate, prior probabilities, and costs of errors. Most existing techniques for adapting ensembles of classifiers require restarting the training, selection, combination, and optimization procedures to account for such a change in operating conditions.



	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
	$BC_{greedy}(LT=\infty)$									
$\lvert E\rvert$	7	10	12	16	18	16	19	21	20	21
$\lvert\mathcal{P}\rvert$	20	40	60	80	100	120	140	160	180	200
	$BC_{greedy}(LT=1)$									
$\lvert E\rvert$	7	7	10	10	9	9	15	15	17	17
$\lvert\mathcal{P}\rvert$	20	27	30	31	38	37	43	40	40	42
	$BC_{greedy}(LT=3)$									
$\lvert E\rvert$	7	7	10	10	12	12	16	16	18	18
$\lvert\mathcal{P}\rvert$	20	40	47	59	57	62	60	66	70	66
	$BC_{greedy}(LT=5)$									
$\lvert E\rvert$	7	7	10	10	12	12	16	16	18	18
$\lvert\mathcal{P}\rvert$	20	40	60	80	87	100	98	104	102	110

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
	$BC_{search}(LT=\infty)$									
$\lvert E\rvert$	5	8	10	9	8	9	8	8	9	12
$\lvert\mathcal{P}\rvert$	20	40	60	80	100	120	140	160	180	200
	$BC_{search}(LT=1)$									
$\lvert E\rvert$	5	5	10	10	3	3	8	8	9	9
$\lvert\mathcal{P}\rvert$	20	25	30	26	29	30	31	31	30	31
	$BC_{search}(LT=3)$									
$\lvert E\rvert$	5	5	8	8	8	8	9	9	8	8
$\lvert\mathcal{P}\rvert$	20	40	46	53	55	56	55	54	54	52
	$B_{search}(LT=5)$									
$\lvert E\rvert$	5	5	8	8	10	10	9	9	8	8
$\lvert\mathcal{P}\rvert$	20	40	60	80	86	97	101	99	95	92

Fig. 16. Illustration of the impact on performance of pruning the pool of HMMs in Fig. 14(a). (a) BC_{greedy} and (b) BC_{search}.

During simulations conducted on both synthetic and real-world HIDS data, the proposed system has been shown to achieve a higher level of accuracy than when parameters of a single best HMM are estimated, at each learning stage, using reference batch and incremental learning techniques. It also outperforms the learn-and-combine approaches using static fusion functions (e.g., majority voting) for combining newly and previously generated pools of HMMs. The proposed ensemble selection algorithms have been shown to provide compact and diverse EoHMMs for operations, and hence simplified Boolean fusion rules, while maintaining or improving the overall system accuracy. The employed pruning strategy has been shown to limit the ever-growing pool size, thereby reducing the storage space for accommodating HMMs parameters and the computational costs for the selection algorithms, without negatively affecting the overall system performance.

The robustness of the proposed ROC-based system depends on maintaining a representative validation data set over time, for selection of HMMs, decision thresholds and Boolean functions. The proposed ADS adopts a human-centric approach, which relies on the system administrator to update the validation set with recent and most informative anomalous sub-sequences. Future work involves investigating techniques such as active learning to reduce labeling and selection costs. The presented results are for validation and test sets with moderate skew (up to 75% of normal and 25% of anomalous data). Another future investigation would involve assessing the impact on system performance of heavily imbalanced data.

Although not explored in this paper, the proposed system can handle concept drift in changing environments, which is typically attributed to changes in prior, class-conditional, or posterior probability distribution of classes [46]. As designed, the ROC-based learn-and-combine approach allows to account for changes in class prior probabilities. Recent techniques for on-line estimation of evolving class priors [65] may be directly employed in the proposed system. This would allow for dynamic selection of the best ensemble of classifiers, decision thresholds and Boolean fusion functions during operations. Other types of drift can be accounted by updating training data and ensemble members, replacing inaccurate classifiers, and adding new features [46]. The ROC-based learn-and-combine approach is designed to select and combine output decisions from any homogeneous or heterogeneous, crisp or soft classifiers, trained on different data or feature subsets according to on-line or batch learning algorithms. Another interesting future work is to evaluate the ability of the proposed system to adapt to concept drift and evolving priors in various detection applications with dynamically changing environments.

Acknowledgments

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada.

References

[1] C. Warrender, S. Forrest, B. Pearlmutter, Detecting intrusions using system calls: Alternative data models, in: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, 1999, pp. 133–45.
[2] X. Hoang, J. Hu, An efficient Hidden Markov Model training scheme for anomaly intrusion detection of server applications based on system calls, in: IEEE International Conference on Networks, ICON, vol. 2, Singapore, 2004, pp. 470–474.
[3] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection for discrete sequences: a survey, Technical Report TR 09-015, University of Minnesota, Department of Computer Science and Engineering, 2009.
[4] S. Grossberg, Nonlinear neural networks: principles, mechanisms and architectures, Neural Networks 1 (1988) 17–61.
[5] R. Polikar, L. Upda, S. Upda, V. Honavar, Learn++: an incremental learning algorithm for supervised neural networks, IEEE Transactions on Systems, Man and Cybernetics, Part C 31 (4) (2001) 497–508.
[6] S.E. Levinson, L.R. Rabiner, M.M. Sondhi, An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition, Bell System Technical Journal 62 (1983) 1035–1074.
[7] L.E. Baum, G.S. Petrie, N. Weiss, A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains, The Annals of Mathematical Statistics 41 (1) (1970) 164–171.
[8] G. Florez-Larrahondo, S. Bridges, E.A. Hansen, Incremental estimation of discrete hidden Markov models based on a new backward procedure, Proceedings of the National Conference on Artificial Intelligence 2 (2005) 758–763.
[9] J. Mizuno, T. Watanabe, K. Ueki, K. Amano, E. Takimoto, A. Maruoka, On-line estimation of hidden Markov model parameters, in: Proceedings of Third International Conference on Discovery Science, DS 2000, vol. 1967, 2000, pp. 155–169.
[10] W. Khreich, E. Granger, A. Miri, R. Sabourin, A comparison of techniques for on-line incremental learning of HMM parameters in anomaly detection, in: Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications (CISDA), Ottawa, Canada, 2009, pp. 1–8.
[11] T. Dietterich, Ensemble methods in machine learning, Multiple Classifier Systems 1857 (2000) 1–15.
[12] L.I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, Wiley, Hoboken, NJ, 2004.
[13] R. Polikar, Ensemble based systems in decision making, IEEE Circuits and Systems Magazine 6 (3) (2006) 21–45.
[14] S. Tulyakov, S. Jaeger, V. Govindaraju, D. Doermann, Review of classifier combination methods, in: H.F. Simone Marinai (Ed.), Studies in Computational Intelligence: Machine Learning in Document Analysis and Recognition, Springer, 2008, pp. 361–386.
[15] G. Brown, J. Wyatt, R. Harris, X. Yao, Diversity creation methods: a survey and categorisation, Journal of Information Fusion 6 (1) (2005) 5–20.
[16] J. Kittler, Combining classifiers: a theoretical framework, Pattern Analysis & Applications 1 (1) (1998) 18–27.
[17] L. Rokach, Ensemble-based classifiers, Artificial Intelligence Review 33 (1) (2010) 1–39.
[18] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140.
[19] Y. Freund, R.E. Schapire, Experiments with a new boosting algorithm, in: ICML 96, 1996, pp. 148–156.
[20] T.K. Ho, J. Hull, S. Srihari, Decision combination in multiple classifier systems, IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (1) (1994) 66–75.
[21] G. Tsoumakas, I. Partalas, I. Vlahavas, An ensemble pruning primer, Applications of Supervised and Unsupervised Ensemble Methods 245 (2009) 1–13.
[22] L.I. Kuncheva, A theoretical study on six classifier fusion strategies, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (2) (2002) 281–286.
[23] T. Fawcett, ROC graphs: Notes and Practical Considerations for Researchers, Technical Report HPL-2003-4, HP Laboratories, Palo Alto, CA, USA, 2004.
[24] F.J. Provost, T. Fawcett, Robust classification for imprecise environments, Machine Learning 42 (3) (2001) 203–231.
[25] W. Khreich, E. Granger, A. Miri, R. Sabourin, Iterative boolean combination of classifiers in the ROC space: An application to anomaly detection with HMMs, Pattern Recognition 43 (8) (2010) 2732–2752.
[26] S. Forrest, S. Hofmeyr, A. Somayaji, The evolution of system-call monitoring, in: Annual Computer Security Applications Conference, 2008. ACSAC 2008, 2008, pp. 418–430.
[27] D. Kershaw, Q. Gao, H. Wang, Anomaly-based network intrusion detection using outlier subspace analysis: a case study, in: C. Butz, P. Lingras (Eds.), Advances in Artificial Intelligence of Lecture Notes in Computer Science, vol. 6657, Springer, Berlin/Heidelberg, 2011, pp. 234–239.
[28] K. Tan, R. Maxion, Determining the operational limits of an anomaly-based intrusion detector, IEEE Journal on Selected Areas in Communications 21 (1) (2003) 96–110.
[29] J. Hanley, B. McNeil, The meaning and use of the area under a receiver operating characteristic (ROC) curve, Radiology 143 (1) (1982) 29–36.
[30] W. Khreich, E. Granger, R. Sabourin, A. Miri, Combining Hidden Markov Models for anomaly detection, in: International Conference on Communications (ICC), Dresden, Germany, 2009, pp. 1–6.
[31] Y.-S. Chen, Y.-M. Chen, Combining incremental hidden Markov model and Adaboost algorithm for anomaly intrusion detection, in: CSI-KDD'09: Proceedings of the ACM SIGKDD Workshop on Cyber Security and Intelligence Informatics, ACM, New York, NY, USA, 2009, pp. 3–9.
[32] L. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proceedings of the IEEE 77 (2) (1989) 257–286.
[33] Y. Ephraim, N. Merhav, Hidden Markov processes, IEEE Transactions on Information Theory 48 (6) (2002) 1518–1569.
[34] A. Dempster, N. Laird, D. Rubin, Maximum likelihood estimation from incomplete data via the EM algorithm, Journal of the Royal Statistical Society, Series B 39 (1) (1977) 1–38.
[35] P. Baldi, Y. Chauvin, Smooth on-line learning algorithms for hidden Markov models, Neural Computation 6 (2) (1994) 307–318.
[36] F. LeGland, L. Mevel, Recursive estimation in hidden Markov models, in: Proceedings of the 36th IEEE Conference on Decision and Control, vol. 4, San Diego, CA, 1997, pp. 3468–3473.
[37] T.K. Ho, The random subspace method for constructing decision forests, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (8) (1998) 832–844.
[38] D.H. Wolpert, Stacked generalization, Neural Networks 5 (1992) 241–259.
[39] F. Roli, G. Fumera, J. Kittler, Fixed and trained combiners for fusion of imbalanced pattern classifiers, Proceedings of the Fifth International Conference on Information Fusion, 2002, vol. 1, 2002, pp. 278–284.
[40] D. Ruta, B. Gabrys, Classifier selection for majority voting, Information Fusion 6 (1) (2005) 63–81.
[41] M. Van Erp, L. Schomaker, Variants of the borda count method for combining ranked classifier hypotheses, in: Seventh International Workshop on Frontiers in Handwriting Recognition, Amsterdam, 2000, pp. 443–452.
[42] D. Ruta, B. Gabrys, A theoretical analysis of the limits of majority voting errors for multiple classifier systems, Pattern Analysis & Applications 5 (4) (2002) 333–350.
[43] N. Littlestone, M.K. Warmuth, The weighted majority algorithm, Information and Computation 108 (2) (1994) 212–261.
[44] K.M. Ali, M.J. Pazzani, Error reduction through learning multiple descriptions, Machine Learning 24 (1996) 173–202.
[45] Å. Raudys, F. Roli, The behavior knowledge space fusion method: analysis of generalization error and strategies for performance improvement, Multiple Classifier Systems 2709 (2003) 55–64.
[46] L.I. Kuncheva, Classifier ensembles for changing environments, Multiple Classifier Systems, Lecture Notes on Computer Science, vol. 3077, Springer-Verlag, Cagliari, Italy, 2004, pp. 1–15.
[47] M. Muhlbaier, A. Topalis, R. Polikar, Incremental learning from unbalanced data, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 2004, pp. 1057–1062.
[48] N.C. Oza, S. Russell, Online bagging and boosting, in: T. Jaakkola, T. Richardson (Eds.), Eighth International Workshop on Artificial Intelligence and Statistics, Morgan Kaufmann, Key West, Florida, USA, 2001, pp. 105–112.
[49] Q. Tao, R. Veldhuis, Threshold-optimized decision-level fusion and its application to biometrics, Pattern Recognition 41 (5) (2008) 852–867.
[50] W. Khreich, E. Granger, A. Miri, R. Sabourin, Boolean combination of classifiers in the ROC space, in: Twentieth International Conference on Pattern Recognition, Istanbul, Turkey, 2010, pp. 4299–4303.
[51] M.J.J. Scott, M. Niranjan, R.W. Prager, Realisable classifiers: improving operating performance on variable cost problems, in: P.H. Lewis, M.S. Nixon (Eds.), Proceedings of the Ninth British Machine Vision Conference, vol. 1, University of Southampton, UK, 1998, pp. 304–315.
[52] J. Neyman, E.S. Pearson, On the problem of the most efficient tests of statistical hypotheses, Royal Society of London Philosophical Transactions Series A 231 (1933) 289–337.
[53] S. Haker, W.M. Wells, S.K. Warfield, I.-F. Talos, J.G. Bhagwat, D. Goldberg-Zimring, A. Mian, L. Ohno-Machado, K.H. Zou, Combining classifiers using their receiver operating characteristics and maximum likelihood estimation, Medical Image Computing and Computer-Assisted Intervention (MICCAI) 3749 (2005) 506–514.

[54] A. Ulaş, M. Semerci, O.T. Yildiz, E. Alpaydin, Incremental construction of classifier and discriminant ensembles, Information Sciences 179 (9) (2009) 1298–1318.
[55] R. Caruana, A. Niculescu-Mizil, G. Crew, A. Ksikes, Ensemble selection from libraries of models, in: ICML'04: Proceedings of the Twenty-First International Conference on Machine Learning, ACM, New York, NY, USA, 2004, p. 18.
[56] L. Rokach, O. Maimon, R. Arbel, Selective voting–getting more for less in sensor fusion, International Journal of Pattern Recognition and Artificial Intelligence 20 (3) (2006) 329–350.
[57] D.D. Margineantu, T.G. Dietterich, Pruning adaptive boosting, in: ICML, 1997, pp. 211–218.
[58] R. Banfield, L. Hall, K. Bowyer, W. Kegelmeyer, A new ensemble diversity measure applied to thinning ensembles, Multiple Classifier Systems 2709 (2003) 306–316.
[59] I. Partalas, G. Tsoumakas, I. Vlahavas, Focused ensemble selection: a diversity-based method for greedy ensemble selection, in: Proceeding of the 2008 Conference on ECAI 2008, IOS Press, Amsterdam, The Netherlands, The Netherlands, 2008, pp. 117–121.
[60] G. Martinez-Munoz, D. Hernandez-Lobato, A. Suraez, An analysis of ensemble pruning techniques based on ordered aggregation, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (2) (2009) 245–259.
[61] W. Fan, F. Chu, H. Wang, P.S. Yu, Pruning and dynamic scheduling of cost-sensitive ensembles, in: Eighteenth National Conference on Artificial Intelligence, American Association for Artificial Intelligence, Menlo Park, CA, USA, 2002, pp. 146–151.
[62] J. Huang, C. Ling, Using AUC and accuracy in evaluating learning algorithms, IEEE Transactions on Knowledge and Data Engineering 17 (3) (2005) 299–310.
[63] S.D. Walter, The partial area under the summary ROC curve, Statistics in Medicine 24 (13) (2005) 2025–2040.
[64] P. Domingos, A unified bias-variance decomposition and its applications, in: Seventeenth International Conference on Machine Learning, Morgan Kaufmann, 2000, pp. 231–238.
[65] Z. Zhang, J. Zhou, Transfer estimation of evolving class priors in data stream classification, Pattern Recognition 43 (9) (2010) 3151–3161.

Wael Khreich is a Ph.D. student in the Laboratoire d'imagerie, de vision et d'intelligence artificielle (LIVIA) at the École de technologie supérieure (ETS). His main research interests are on-line and incremental learning for stochastic models such as hidden Markov models, and decision fusion in multi-classifier systems, with applications in intrusion detection in computer and network security.

Eric Granger obtained a Ph.D. in Electrical Engineering from the École Polytechnique de Montréal in 2001, and from 1999 to 2001, he was a Defence Scientist at Defence R&D Canada in Ottawa. Until then, his work was focused primarily on neural network signal processing for fast classification of radar signals in Electronic Surveillance (ES) systems. From 2001 to 2003, he worked in R&D with Mitel Networks Inc. During that time, he designed algorithms and dedicated electronic circuits (ASIC/SoC) to implement cryptographic functions in Internet Protocol (IP)-based communication platforms. In 2004, Dr. Eric Granger joined the ÉTS, where he has been developing applied research activities in the areas of machine learning, patterns recognition, artificial neural networks, signal processing and microelectronics. He presently holds the rank of Assistant Professor in the département de génie de la production automatisée (GPA). Since joining ÉTS, he has been a member of the Laboratoire d'imagerie, de vision et d'intelligence artificielle (LIVIA), and his main research interests are adaptive classification systems, incremental learning, ambiguity and novelty detection, neural and statistical classifiers, and multi-classifier systems, with applications in military surveillance (recognition of radar signals), biometric authentification (recognition of individuals from their signatures and their faces), and intrusion detection in computer and network security.

Ali Miri is a Professor with the School of Computer Science of Ryerson University, Toronto, Canada. He is also a Professor at the School of Information Technology and Engineering, and the Department of Mathematics and Statistics at the University of Ottawa, Ottawa, Canada. His research interest include applied cryptography, digital communication, signal processing, distributed systems, and mobile computing. He is a member of Professional Engineers Ontario, ACM and a senior member of IEEE.

R. Sabourin joined in 1977 the Physics Department of the Montreal University where he was responsible for the design, experimentation and development of scientific instrumentation for the Mont Mégantic Astronomical Observatory. His main contribution was the design and the implementation of a microprocessor-based fine tracking system combined with a low-light level CCD detector. In 1983, he joined the staff of the École de Technologie Supérieure, Université du Québec, in Montréal where he co-founded the Department of Automated Manufacturing Engineering where he is currently Full Professor and teaches Pattern Recognition, Evolutionary Algorithms, Neural Networks and Fuzzy Systems. In 1992, he joined also the Computer Science Department of the Pontifícia Universidade Católica do Paraná (Curitiba, Brazil) where he was co-responsible for the implementation in 1995 of a master program and in 1998 a Ph.D. program in applied computer science. Since 1996, he is a senior member of the centre for Pattern Recognition and Machine Intelligence (CENPARMI, Concordia University). Dr. Sabourin is the author (and co-author) of more than 260 scientific publications including journals and conference proceeding. He was co-chair of the program committee of CIFED'98 (Conférence Internationale Francophone sur l'Écrit et le Document, Québec, Canada) and IWFHR'04 (9th International Workshop on Frontiers in Handwriting Recognition, Tokyo, Japan). He was nominated as Conference co-chair of ICDAR'07 (9th International Conference on Document Analysis and Recognition) that has been held in Curitiba, Brazil in 2007. His research interests are in the areas of handwriting recognition, signature verification, intelligent watermarking systems and bio-cryptography.