Benchmarking Evolutionary Computation Approaches to Insider Threat Detection

Duc C. Le
Dalhousie University
Halifax, NS, Canada

Sara Khanchi
Dalhousie University
Halifax, NS, Canada

A. Nur Zincir-Heywood
Dalhousie University
Halifax, NS, Canada

Malcolm I. Heywood
Dalhousie University
Halifax, NS, Canada

ABSTRACT

Insider threat detection represents a challenging problem to companies and organizations where malicious actions are performed by authorized users. This is a highly skewed data problem, where the huge class imbalance makes the adaptation of learning algorithms to the real world context very difficult. In this work, applications of genetic programming (GP) and stream active learning are evaluated for insider threat detection. Linear GP with lexicase/multi-objective selection is employed to address the problem under a stationary data assumption. Moreover, streaming GP is employed to address the problem under a non-stationary data assumption. Experiments conducted on a publicly available corporate data set show the capability of the approaches in dealing with extreme class imbalance, stream learning and adaptation to the real world context.

CCS CONCEPTS

• **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**; • **Theory of computation** → *Genetic programming*; *Online learning algorithms*;

KEYWORDS

Insider threat detection, Cyber-security

ACM Reference Format:
Duc C. Le, Sara Khanchi, A. Nur Zincir-Heywood, and Malcolm I. Heywood. 2018. Benchmarking Evolutionary Computation Approaches to Insider Threat Detection. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205612

1 INTRODUCTION

Insider threat detection represents a particularly challenging example of an intrusion detection task. It is not possible to assume that 'walls' or 'barriers' can be built between an outside source of malicious behaviour, because the source(s) of such behaviour are on 'inside' the organization. Moreover, the malicious behaviour(s) are not necessarily static, and can change over time or might develop over a period of time. Finally, in most organizations the individuals with malicious intent represent an infrequently occurring behaviour, thus data available to describe the activity is particularly rare. This work represents an initial study in which we make use of the CERT insider thread dataset (Section 4) to benchmark different machine learning (ML) approaches, divided into two basic categories. The categories are selected to reflect different basic underlying assumptions regarding access to data.

Category 1 algorithms assume that the ML models can be built *offline* under the traditional supervised learning stationary data assumption i.e., training data is completely representative of the overall task, and models are built once and deployed on an independent test partition. Category 2 algorithms assume that ML models can be built *online* by interacting with the data as a stream generated by a non-stationary process. Moreover, it is not possible to label all the data, so such algorithms have to operate under a label budget that defines how much of the data an expert can be called on to label [27]. Algorithms of this class therefore have to answer additional questions such as what to request labels for, and how much of the material that has been labeled can be retained, i.e. it is not possible (or desirable) to save everything because the underlying task might also be non-stationary. Finally, we note that malicious behaviours could also be adopted that introduce 'adversarial samples' into the stream, thus aiming to fool the ML algorithm into characterizing malicious behaviour as normal [1, 25]. In order to address this latter point, we assume that the human expert needs to provide the label, ground truth, but does not decide what to label.

In the following, Section 2 summarizes previous research in the general area of Insider threat detection. Section 3 introduces the GP frameworks later used for benchmarking, as well as the non-evolutionary ML algorithms. Section 4 details how the CERT data set expresses the Insider threat problem and how features are then derived. Parameterizations for the GP algorithms are also discussed and performance metrics established. Section 5 presents the performance evaluations under each application constraint category, with concluding remarks made in Section 6.

2 BACKGROUND

The potential utility of adopting a streaming approach to insider threat detection problem was first recognized in [5, 17]. Multiple algorithms were applied to the KDD'99 dataset. The principal limitation of the evaluation was that the KDD'99 dataset pertains to intrusion detection as a whole as opposed to insider detection in particular. More recently, Parveen *et al.* assumed an incremental learning approach to insider threat detection under streaming data [18]. The stream is divided into non-overlapping data chunks, and

GECCO '18, July 15–19, 2018, Kyoto, Japan

a quantized dictionary of patterns in data is created based on data from the current data chunk and some of the most recent chunks using a weighted sum scheme. Test data is considered an anomaly if it has large edit distance from all patterns in the dictionary. Such a scheme is potentially capable of detecting threats across multiple users. An alternative approach to insider threat detection with machine learning is to adopt a representation explicitly capable of expressing temporal properties. Rashid *et al.* use Hidden Markov Models (HMMs) to capture each user's normal weekly activity sequences and detect the deviation that may potentially indicate insider threats [19]. The model requires training one HMM per user per week on explicitly 'normal' data. Their results illustrate the potential of HMM as an unsupervised learning method for anomaly detection in general and insider threat detection in particular. The temporal nature of insider threats can also potentially be captured using a recurrent neural network. A combination of a deep neural network and recurrent neural network were used to recognize each user's behaviours and output anomaly scores [23]. As per the HMM, one model is required per user, and the model needs to be retrained at each time step. The scores for recent time steps (days) are combined using weighted moving average. Experiments are performed on r6.2 of CERT insider threat dataset, and Cumulative recall is used for measurement. They argue that *"the cost of a missed detection is substantially larger than the cost of a false positive, we feel that recall-oriented metrics such as CR-k are a more suitable measurement of performance than precision-oriented ones"*; thus performance is reported in terms of recall alone. Rather than have the ML explicitly capture temporal properties in the original user data, features might be crafted by experts to capture such information. This also implies that algorithms designed for anomaly detection and/or supervised learning can be applied for insider threat detection. Such an approach was adopted by [21].

3 LEARNING ALGORITHMS

3.1 Linear GP

In this paper, Linear genetic programming (LGP) is employed as one of the supervised learning methods for classifying data to detect insider threats under a stationary data assumption. LGP is a variant of GP where programs in a population are represented in linear form, as a sequence of instructions from an imperative programming language [3]. Each instruction executes an operation over the operands, which can be registers, constants, or input value. Then result of each instruction is stored in a register. The final result of the program is taken as the values of the registers which are designated as the output registers at the end of the program.

There are two properties that differentiate LGP from other representations of GP. First, the imperative representation of LGP allows the data to be processed as in a directed graph data flow, thus facilitating reuse of register content by multiple instructions. This in turn allows the reuse of subprograms for evolving compact solutions. Second, structurally noneffective code (introns) in LGP programs - instructions that have no impact on the output registers - support neutral variation and skipping of intron code during fitness evaluation, where noneffective code can be tuned effective by variation operators.

Each generation, LGP evolves the program population by determining fitness for each program in the population for a subset of training data. Variation operators, mutation and crossover, introduce new material for consideration at the next generation. Crossover in LGP is done similarly to a genetic algorithm, blocks of genes (representing instructions) are swapped between pairs of parents. Mutation operates at two levels: macro mutation, impacts an entire instruction (i.e. instruction replacement, deletion or insertion), and micro mutation, where a field of a current instruction is effected (target register, operands, or operator).

Selection operators determine which individuals in the population are selected to generate the next population. In this study, as there are inherently uncompromising objectives of the insider threat detection, that are high malicious detection rate and low false positive rate. Hence, for that purpose, two selection methods - lexicase [10] and multi-objective selection - are employed in this study. Parent selection in the lexicase selection algorithm follows a randomized order of test cases each time. On the other hand, multi-objective selection in this study is based on Pareto-ranking with the two main objectives: class-wise detection rate and accuracy are taken into account.

3.2 GP Teams

Teaming formulations of genetic programming (hereafter GP teams) enable task decomposition to take place across multiple programs, or co-operative coevolution [15]. Recent research has shown that adopting a teaming formulation for GP is particularly beneficial under the context of streaming classification tasks with non-stationary data [11, 24]. In particular, change is not necessarily associated with all of the task changing, but instead change might be reflected in updates to different aspects of specific classes at different points in time. Hence, in adopting a modular approach to the streaming task, it is possible to more quickly react to changes in the data [24].

The approach adopted to GP teams in this work takes the form of Symbolic bid-based GP (or SBB) [16]. Such a framework co-operatively coevolves a population of programs and a population of candidate teams. The population of teams assumes a variable length representation in which each individual is a set of pointers to (a unique subset of) individuals in the program population. This means that team complement is evolved (it is not necessary make assumptions regarding team membership), and multi-class classification appears as an emergent property of team complement. The only constraints are that each member of the team population have to index at least two members of the program population, and there should be at least two different classes represented by each team.

Members of the program population assume a LGP representation (Section 3.1). However, each program is also associated with *one* scalar class label, $a \in C$, where C is the set of class labels. Thus, the program output represents the 'certainty' for its corresponding label. Now, consider a team consisting of two programs, p_a and p_b. Both programs are evaluated on the current exemplar, producing a single output for each program, $p_a.out$ and $p_b.out$. Which ever of the two programs has the maximum output 'wins' the right to suggest its class label, a. If a matches the actual class label, then the fitness of the team improves. Thus, fitness is only directly expressed at the level of teams.

SBB assumes a breeder formulation for evolution, thus at each generation the worst $Tgap$ teams are deleted. At this point the program population is tested for any programs which are not indexed by a team. If any such programs exist, they must have been associated with the worst performing teams, and are therefore also deleted. Selection begins at the team population with $Tgap$ teams selected as parents with uniform probability, and begins by cloning the parents. Variation operators probabilistically add and/or delete programs from the selected parent teams. Likewise, a program can be cloned and variation applied to the program as per LGP.

3.3 Streaming GP

As introduced in Section 1, a GP teaming formulation will be assumed for operation under the streaming data scenario (Section 3.2). However, GP teaming does not address how GP 'interfaces' with the stream such that operation under a label budget is possible. Figure 1 summarizes the general approach, where this has previously been evaluated under various non-stationary streaming data benchmarks with artificial data [13, 14].

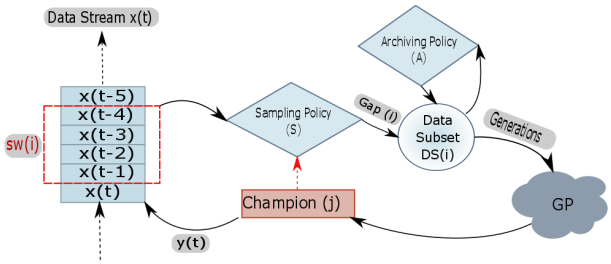


Figure 1: Stream GP framework

A non-overlapping window provides the initial interface to the stream. At each time, t, only the content of current window, $SW(i)$ is available and accessible for making label queries. The Data Subset, $DS(i)$, represents exemplars for which the true label information has been requested. After each update to the Data Subset, τ generations of GP are performed, so identifying a champion classifier, gp^*, which provides label predictions on an 'anytime' basis. Thus, after a cold start, a gp^* individual is always available for label prediction *before* label requests are made. Evolution of new champion classifiers takes place throughout the stream at a rate of replacement dictated by the label budget β.

Sampling and Archiving policies determine what to sample/retain in the Data Subset. Following the recommendations of Khanchi *et al.* [13, 14], the **Sampling policy (S)** selects up to Gap instances from current window location, $SW(i)$, with a uniform p.d.f. Such a sampling process does not make use of label information. The size of the GP non-overlapping window reflects the label budget β. As $\beta \rightarrow 1(0)$ window size decreases to Gap (increases to ∞), implying that every exemplar has its label requested (never requested), or $\beta = \frac{Gap}{DS_{size}}$.

True label information is provided by a human expert and placed in the data subset. This guarantees the validity of label information, so avoiding the pathology of adversarial samples [1, 25]. The **Archiving policy (A)**, on the other hand, has the true label information, incrementally balances the Data Subset class distribution by targeting exemplars from the over-represented classes for replacement. The combination of these Archiving and Sampling policies leads to an incrementally balanced data subset in which valuable minor classes are retained [13, 14].

3.4 Comparator algorithms

Two of the most popular ML algorithms – Decision tree and Bayesian based classifiers – provide the basis for comparison against GP in this study [26]. This provides an example of each algorithm for stationary and non-stationary training scenarios, using the implementations in WEKA and MOA software suites [2, 6].

3.4.1 Decision trees. Decision trees are notable for its popularity and interpretability. Under the **stationary data** assumption, a decision tree is generated using C4.5 algorithm [26]. C4.5 is based on the concept of information entropy and creates an if-then rule at each decision tree node. A normalized information gain measure is calculated at each tree node and splits the data into subsets, such that the 'purity' of a subset increases as the tree is descended. The Hoeffding tree algorithm is employed on **non-stationary data**. The algorithm for constructing a Hoeffding tree progressively splits leaves and creates decision nodes where enough statistics are available at the leaf. For each exemplar, it traverses from the root node to a leaf and updates the statistics along the way. The decision on when and how to split the leaf is based on the hoeffding bound [12] which with a certain confidence suggests an attribute for splitting. The Hoeffding tree has been successfully applied on multiple streaming applications [7] and generalized to operate under label budgets [27].

3.4.2 Bayesian based algorithms. On **stationary data**, Bayesian networks have been shown to be capable of working in other cybersecurity applications [9, 22]. A Bayesian network describes a directed graphical model representing a set of features and their relationships using Bayesian probabilities. Each node in the network represents a data attribute, and the directed link between nodes represents the conditional dependency between the attributes. For classification applications, a Bayesian network models the relationship between the features and class labels.

On **non-stationary data**, Naive Bayes is applied to streaming applications incrementally [20]. The learning process starts with an initial data subset and estimates the prior probability ($P(C)$) and conditional probability ($P(X|C)$). Then, as the stream progresses, new data subsets are prepared and these two parameters are updated based on the new information. At anytime the label prediction for stream exemplars is done based on the available parameters and calculating the post probability ($P(C|X)$). It is noteworthy that Bayesian based algorithms work on discrete attributes, which makes it a promising approach for the data in this study. Moreover, a generalization to the case of operation under label budgets is readily available [27].

Table 1: CERT dataset exemplar counts. There are 5 classes: class 1 denotes normal user behaviours, classes 2-4 denote three different types of insider behaviours, class 5 represents in part malicious behaviours by class 4 users using other user's accounts

	Class 1	Class 2	Class 3	Class 4	Class 5
CERT-Weekly	48041	254	52	10	10
CERT-Daily	233557	1252	265	52	50

Table 2: Stream GP Parameters.

Parameter	Value
Data Subset size (DS_{size})	120
DS gap size (Gap)	20
GP gap size ($Tgap$)	20
Team pop. size (P_{size})	120
Max. programs per host (ω)	20
Prob. Program deletion (pd)	0.3
Prob. Program addition (pa)	0.3
Prob. Action mutation (μ)	0.1

4 EMPIRICAL METHODOLOGY

4.1 Dataset and feature extraction

Obtaining data for designing and evaluating insider threat detection systems encounters several additional difficulties unique to the application area. Insider threats typically involve corporations and government agencies, where such threats relate to compromising organizational intellectual properties. Hence the data is usually not made available to research community.

This study employs the publicly available CERT insider threat dataset[1] [8], one of a very few datasets available for benchmarking purposes. A number of different model types including communication and connection graph models, topic models, behavioural models, and psychometric models are employed for generating the data as close to what is seen in the real-world as possible. Furthermore, the insider threat data provided is synthesized in the same form and scope as the normal data. There are a total of 5 insider threat scenarios, ranging from data leaking, intellectual property theft to IT sabotage.

The dataset is divided into multiple releases, each of which characterize an organization with 1000 employees over a time period of either a day or a week. Data in each release describes two properties: (i) the users' activity logs, which includes log on, device, file, email, and http logs, and (ii) the organization's structure and users' information in a Light-weight directory access protocol (LDAP) folder. For evaluating the algorithms, raw data from the two are combined, additional models, such as user - host are built, before numerical features are extracted.

Two categories of attributes appear: user features and activity features. User features include each user's role, functional unit, department, psychometric scores, and employment status. The activity features mostly take the form of specific activity count in each log file over a given time period, such as: `number of logins`, `number of logins in after hours`, `number of external emails`, `number of cloud storage visits`. Moreover, two versions of the CERT insider threat dataset are extracted based on the extraction window: weekly and daily data. While weekly data may capture a more complete picture of a user's activities over a longer time period (both week days and weekend days are taken into account), daily data on the other hand may allow better responsiveness of a detection system, i.e. insider threat is detected earlier.

In this paper, release 4.2 of the dataset, hereafter the 'CERT' dataset, is employed for designing and evaluating insider threat detection approaches. Release 4.2, which spans over 72 weeks, contains a significantly greater amount of insider threat incidents than other releases. This provides the basis for testing the proposed detection systems against a more diverse set of scenarios. Given the nature of the task, an extreme imbalance exists in the distribution of classes (Table 1). With this in mind, since there are no malicious behaviours in the first 20 weeks, the actual data employed in this work is from week 20 onwards of original CERT dataset. This decision is made to evaluate some of the issues with class imbalance. Moreover, from the perspective of streaming classifiers, there may very well be shifts and drifts in a user's behaviours and activities over the duration of the datasets.

[1] https://www.cert.org/insider-threat/tools/index.cfm

4.2 Parameterization

The GP model deployed with batch updating in this study is an implementation of Linear GP [3], where each program is defined in terms of 1 and 2 argument operations (`+, -, x, /, sin, cos, sqrt, log, exp, if`) with operands taking the form of either registers or constants. Post execution, the program's output class is defined by the output register with maximum value. As noted in Section 3.1, we compare two selection methods: Lexicase selection [10] and a Pareto-ranking multi-objective (overall detection rate and accuracy). The main parameters of GP are: population size = 2500, mutation prob. = 0.8, crossover prob. = 0.4, number of generations = 2000, tournament size = 5.

Stream GP applies the same parameters as previous work [14] which is represented in Table. 2. At each training epoch, $Gap = 20$ exemplars from data subset and $Tgap = 20$ teams from team population are replaced and $\tau = 5$ GP generations are performed. The experiments are done under two different label budgets, $\beta = 0.05, 0.25$. The same label budget is used for Naive bayes and Hoeffding Tree streaming policies.

4.3 Performance metrics

In this work, performances of classifiers based on a stationary learning approach is measured as traditionally based on detection rate (DR) of each class and accuracy on the testing data, Eq. (1):

$$DR_j = \frac{tp_j}{tp_j + fn_j}, \text{ and } Accuracy = \frac{\sum_j tp_j}{\sum_j tp_j + \sum_j fn_j}, \quad (1)$$

where tp_j and fn_j are the counts of true positive and false negative for class j post testing. On the other hand, given the nature of training data in stream learning algorithms, the performances need

to be measured dynamically throughout the stream. Hence, the same measures as in Khanchi et al. [13, 14] are applied in this paper. Thus, in this case the detection rates are incrementally estimated as a function of time, t, as well as class. A multi-class detection rate $DR(t)$ is used to measure the performances of non-stationary learning algorithms, Eq. (2):

$$DR(t) = \frac{1}{C}\sum_{j=1}^{C} DR_j(t), \tag{2}$$

where C represents the number of data classes encountered up to t and the Detection rates of majority and minority classes contribute equally to $DR(t)$. Thus, $DR(t)$ allows a fair comparison between stream learning algorithms under the imbalanced data condition, where all classes are taken into account. Finally, we note that Eq. (2) describes Detection rate as a function of time, hence can also be summarized as a single scalar 'area under the curve' [11].

5 EXPERIMENT RESULTS

5.1 Static offline classifier evaluation

The algorithms are benchmarked on the CERT dataset using two scenarios for splitting the training–testing data: 1) the first 50% of the data is used for training, and the last 50% for test, and 2) 50% of the data is sampled uniformly from the *entire* dataset for training (with the remainder representing the test partition). Naturally, the first scenario is more representative of a deployment situation, as it implies a detection model has to be built first before deployment. The second scenario provides an indication of how much better the models might be if it was possible to sample across the entire detection period.

Results from LGP with both lexicase and multi-objective selection are estimated over an average of 4 different runs in each experiment. For C4.5 and Bayesian Network, when the first 50% of data provides the training partition, the results are taken from a single run, i.e. C4.5 and Bayes Net return the same results for the same data. When 50% of the data are sampled randomly for training, the results are the average of 20 independent runs (effect of different data samples).

The results of the stationary/non streaming algorithms on both weekly and daily CERT data are summarized in Table 3. Note that this reflects a binary classification task, where the two classes are normal (class 1) and insider threats (classes 2-5). It appears that under a stationary data assumption, the supervised learning algorithms employed in this study generally perform well and give stable results. Most of the algorithms achieve higher than 90% in both class-wise detection rate and accuracy. Additionally, we observe that the results represent a trade-off between accuracy (normal DR) and insider threat detection rates. This can be explained by the extremely imbalanced class distribution in the data.

The LGP algorithms achieve high insider detection rate with reasonable false positive rate in most cases. Interestingly, LGP with multi-objective selection usually evolve simpler solutions (shorter solution length). Naturally, the cost of false negatives (miss classifying a threat as normal) is higher than the cost of false positives. However, given the cost of an analyst performing follow up on cases characterized as a threat, it is also important to minimize the false positives. The results indicate that C4.5 struggles to find the patterns to detect insider threats in all cases. The possible explanation is that C4.5 focuses on achieving the highest accuracy, hence is not able to learn from the minority class well. Finally, Bayesian network achieve similar results to LGP in most of the cases.

Over the two versions of the dataset, while C4.5 performance is largely unchanged, the different selection methods of LGP shows an interesting observation. On CERT-weekly data, LGP with lexicase selection performs better than multi-objective LGP, i.e. higher accuracy via higher normal DR and maintains the similar insider DR. However, on the CERT-daily task, the trend is reversed, i.e. multi-objective LGP gives better normal DR, at a cost of lower insider DR. Bayesian network clearly achieves the best balance between accuracy and class-wise detection rate in CERT-daily dataset, while LGP with lexicase selection is the best performer on weekly data.

Finally, on the two training-testing data split approaches, generally the experimental results are better where 50% of data is randomly selected for training. The reason behind this is likely that temporal variation exists in user behaviours. Given the long duration of the dataset, models trained on first part of the data are likely not capable of adapting well to the user behaviours in the second part of the dataset.

5.2 Streaming online algorithm evaluation

The Stream GP algorithm (*Archive*) and two MoA algorithms, *Hoeffding Tree* and *Naive Bayes* with variable uncertainty sampling are evaluated. A total of 20 trials are performed per algorithm independently at two different label budgets, $\beta = \{0.05, 0.25\}$. In all cases, classification models are incrementally constructed over the course of the dataset (stream), implying that the model at the end of the stream is potentially very different from that at the beginning of the stream. In order to provide an impression for how much might be gained by seeing the same data more than once, we actually present the same data twice. Thus, the 50% point represents one 'pass' through the CERT dataset. The remaining 50% of the stream represents a 'second pass' through the CERT dataset. During this second pass, the models experience a second opportunity to sample the *same* data, i.e. a label budget of 2β is simulated during the second pass.

5.2.1 Overall Performance. The Friedman non-parametric repeated measures statistic is employed to showcase the trend of algorithms on the four combinations of the data set [4]. Tables 4 and 5 summarize the results by ranking the algorithms. The last column represents the average rank of each algorithm from which the Friedman statistic is estimated [4]. The result of the Friedman test χ_F^2 for $\beta = \{0.05, 0.25\}$ is 8 and 6.5 respectively, where the critical value of $F(2, 6)$ for $\alpha = 0.05$ is 6, so the null-hypothesis is rejected in each case. Applying the Nemenyi post hoc test groups algorithms with equivalent performance. If the average rank is within the critical value of $CD = 1.6$ for $q_\alpha = 0.05$ then they are considered equivalent. In this case, stream GP performs significantly better than Hoeffding Tree, but there is insufficient data to reach a conclusion regarding Naive Bayes.

5.2.2 Class-wise detection rate. In order to give a better perspective of the algorithms behaviour throughout the course of the stream, the DR metric can be plotted over the course of the

Table 3: Results of non streaming algorithms

Algorithm	Train on first 50% of data				Train on random 50% of data			
	Normal DR	Insider DR	Accuracy	Solution length	Normal DR	Insider DR	Accuracy	Solution length
CERT-Weekly								
LGP - Lexicase	97.42	88.50	97.37	122	97.81	89.08	97.77	75
LGP - MultiObj	92.70	88.50	92.68	16	88.00	93.67	88.04	19
C4.5	99.88	47.79	99.62	49	99.88	62.09	99.64	79
Bayes Net	89.49	96.52	89.52		88.38	92.88	88.41	
CERT-Daily								
LGP - Lexicase	84.84	91.31	84.87	26	90.60	89.39	90.59	101
LGP - MultiObj	93.40	82.23	93.37	44	95.64	85.51	95.60	53
C4.5	99.98	42.77	99.87	68	99.98	77.86	99.89	117
Bayes Net	96.25	87.65	96.23		94.90	88.30	94.87	

Table 4: Algorithm ranks w.r.t. streaming average DR metric under a 25% label budget. Bracketed entries represent median DR values to 1 decimal place. Naive Bayes (NB) and Hoeffding tree classifiers (from MoA) appear with 'variable' sampling protocol. SBB algorithm is 'Archive' which uses uniform sampling.

Data set	CERT-Weekly		CERT-Daily		R_j
	Binary DR	Multi-class DR	Binary DR	Multi-class DR	
stream GP	**1 (78.8)**	**1 (53.4)**	**1 (81.3)**	**1 (56.9)**	**1**
Hoeffding	3 (52.3)	3 (20.0)	2.5 (57.5)	2.5 (29.3)	2.75
Naive bayes	2 (59.2)	2 (24.1)	2.5 (57.5)	2.5 (29.3)	2.25

Table 5: Algorithm ranks w.r.t. streaming average DR metric under a 5% label budget. Bracketed entries represent median DR metric values to 1 decimal place. Naive Bayes (NB) and Hoeffding tree classifiers (from MoA) appear with 'variable' sampling protocol. GP streaming algorithm is 'Archive' which uses uniform sampling.

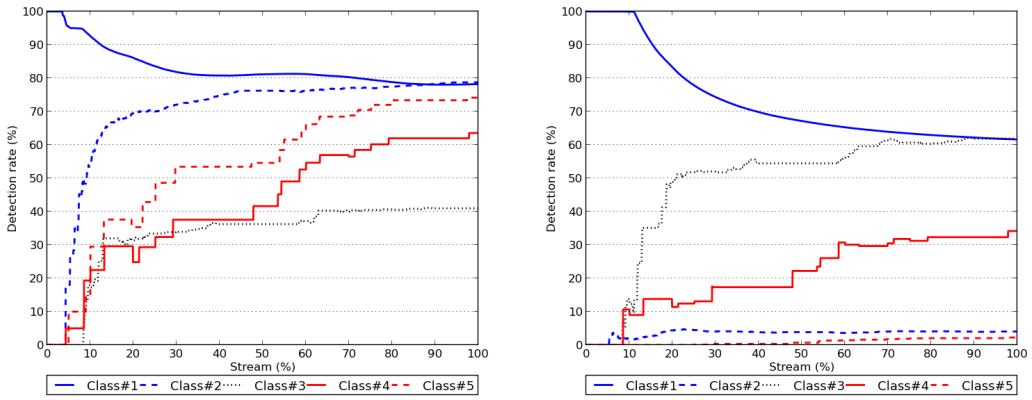
Data set	CERT-Weekly		CERT-Daily		R_j
	Binary DR	Multi-class DR	Binary DR	Multi-class DR	
stream GP	**1 (67.8)**	**1 (33.3)**	**1 (74.1)**	**1 (39.8)**	**1**
Hoeffding	3 (50.0)	3 (19.9)	3 (50.8)	3 (20.1)	3
Naive bayes	2 (59.6)	2 (22.7)	2 (53.0)	2 (24.1)	2

stream. We are interested to know if stream GP and Naive Bayes (identified with the best overall performance, Section 5.2.1) behave similarly throughout the stream. Figure. 2 compares the performance of stream GP and Naive Bayes over the CERT-Daily data set for $\beta = 0.25, 0.05$.

As noted above, the 50% interval on the x-axis denotes one pass through the dataset, after which the 50-100% interval represents a second pass. Given the label budget parameterization, this implies that by the end of the second pass the $\beta = 0.25$ parameterization has encountered a total label budget of 50%. These models are also deployed as multi-class classifiers, thus we can also inspect to what degree different classes are detected (class 1 is the only non-malicious user).

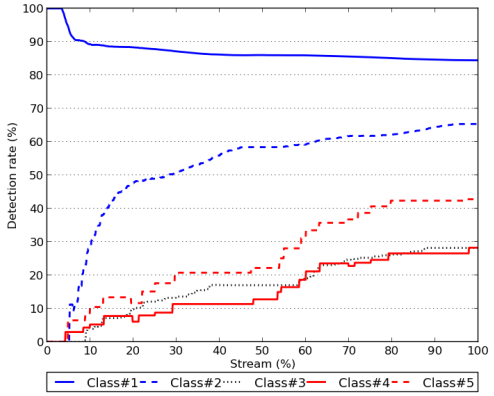
Figure 2(a) summarizes the average Detection rate per class over the 20 runs for each algorithm. Both stream GP and Naive Bayes begin by labeling everything as the major class. After about the first 10% of the stream, detection of class 1 decreases somewhat as the detection of the remaining (malicious) classes improves. Both algorithms see a more pronounced decrease in class 1 detection under the $\beta = 0.25$ label budget, but also a far stronger detection of the malicious classes. It is also interesting to note that stream GP is much better at detecting classes 2, 4 and 5, whereas Naive Bayes detects class 3 better than stream GP. Under the $\beta = 0.05$ label budget, stream GP appears to perform better than Naive Bayes in all but class 3.

5.2.3 Computational cost. Computational cost for a real-time problem should be reasonably low which GP is not known for. Here, we demonstrate that the stream GP framework is able to provide 'near' real-time operation. Wall clock time is calculated based on two aspects: 1) the cost of performing fitness evaluation, and 2) the cost of deploying the champion to make predictions for new

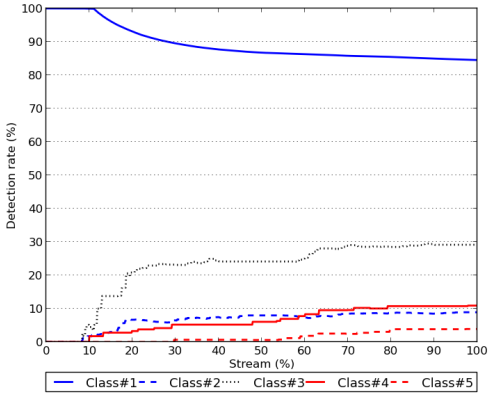


(a) Stream GP (25%)

(b) NB Variable (25%)



(c) Stream GP (5%)



(d) NB Variable (5%)

Figure 2: CERT Daily class-wise Detection rate through the stream. 25% versus 5% label budget. Class 1 is Normal, class 2-5 represent insider threat attacks (ordered).

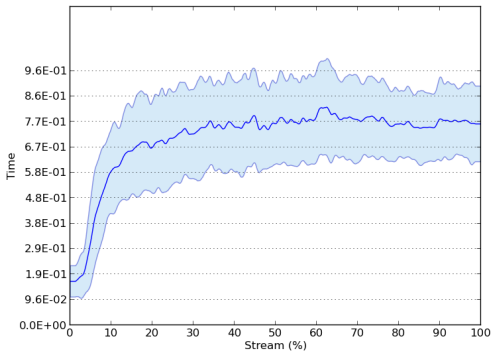
data (champion labeling). These timings can be viewed in Figure. 3. Naturally, the time taken by the human expert to suggest true labels for data under the label budget does not appear. However, there is always a champion available to provide label prediction at anytime, meanwhile the expert can continue to provide true labels. In practice, this would insert a lag in the provision of label information, where such lags are always present in supervised learning systems. The times are the average of 20 *single-threaded* runs on Intel i5 CPU 16@2.67GHz and 48GB memory.

The average time for the champion to suggest labels for a window is $\approx 7.6\mu$s and for the fitness evaluations is 0.7s. The fitness evaluation time is calculated based on the time it takes to perform $\tau = 5$ training epochs on a window plus the champion selection. The number of evaluations is calculated as $DS = 120$ exemplars for each window that should be learned by $P_{size} = 120$ teams. This number is reduced to learning new exemplars ($Gap = 20$) and updating the new Teams ($T_{gap} = 20$) with the data subset or $20 \times 20 \times 5$ evaluations. In short, the framework is sufficient to operate in real time, with a lag relative to the expert's ability to provide labels.

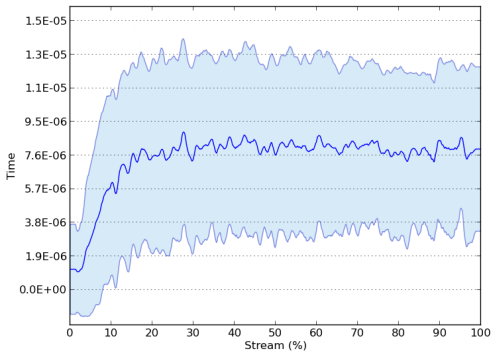
Finally, from an insider threat detection application perspective, we note that where each data instance represents a day or a week of a user's activities, thus ground truth can potentially be provided by an analyst with such small label budgets (5% or 25%).

6 CONCLUSIONS

In this paper, GP algorithms in two categories, static and active learning, are benchmarked against popular machine learning algorithms in the literature for the corporate insider threat detection task, under two data assumptions, stationary and non-stationary. The insider threat detection task represents a challenging problem, where class distributions are highly imbalanced, ground truth may be limited, and shifts and drifts may appear in user behaviours over time. Results show that the GP algorithms achieve satisfactory performance, with good balances between insider threat detection rates and normal data misclassification rates. More specifically, the stream active learning GP outperforms the comparing algorithms in all experiments. Moreover, the results are achieved at only a small computational cost. This demonstrates the capabilities of the approach in dealing with real-world and real-time active learning



(a) Fitness evaluation



(b) Champion labeling

Figure 3: Wall clock time for of stream GP on CERT-daily data set (multi-class version)

problems. Future work will investigate the active learning approach under more extreme conditions, such as much higher class imbalance in other versions of the CERT dataset. Given the nature of the insider threat detection task, where the ground truth may be hard to obtain, an approach where GP algorithms act as an anomaly detection system on a change detection basis, or are teamed with other anomaly detection algorithms can be considered.

ACKNOWLEDGMENTS

This research is supported by Natural Science and Engineering Research Council of Canada (NSERC). Duc C. Le gratefully acknowledges the support by Killam trust, Mitacs, and Nova Scotia Research and Innovation funding programs.

REFERENCES

[1] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar. 2010. The security of machine learning. *Machine Learning* 81, 2 (2010), 121–148.
[2] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. 2010. MOA: Massive Online Analysis. *Journal of Machine Learning Research* 11 (2010), 1601–1604.
[3] M. F. Brameier and W. Banzhaf. 2007. *Linear Genetic Programming*. Springer US.
[4] J. Demsar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research* 7 (2006), 1–30.
[5] W. Eberle, L. Holder, and D. Cook. 2009. Identifying Threats Using Graph-based Anomaly Detection. In *Machine Learning in Cyber Trust*. Springer, 73–108.
[6] F. Eibe, M. A. Hall, and I. H. Witten. 2017. The WEKA Workbench. In *Data mining: practical machine learning tools and techniques* (4 ed.). Morgan Kaufmann.
[7] J. Gama. 2012. A survey on learning from data streams: current and future trends. *Progress in AI* 1, 1 (2012), 45–55.
[8] J. Glasser and B. Lindauer. 2013. Bridging the Gap: A Pragmatic Approach to Generating Insider Threat Data. In *IEEE Symposium on Security and Privacy Workshops*. 98–104.
[9] F. Haddadi and A. N. Zincir-Heywood. 2015. A Closer Look at the HTTP and P2P Based Botnets from a Detector's Perspective. In *Foundations and Practice of Security - 8th International Symposium (FPS 2015)*. Clermont-Ferrand, France, 212–228.
[10] T. Helmuth, L. Spector, and J. Matheson. 2015. Solving Uncompromising Problems With Lexicase Selection. *IEEE Transactions on Evolutionary Computation* 19, 5 (2015), 630–643.
[11] M. I. Heywood. 2015. Evolutionary model building under streaming data for classification tasks: opportunities and challenges. *Genetic Programming and Evolvable Machines* 16, 3 (2015), 283–326.
[12] G. Hulten, L. Spencer, and P. M. Domingos. 2001. Mining time-changing data streams. In *ACM SIGKDD International Conference on Knowledge discovery and data mining*. 97–106.
[13] S. Khanchi, M. I. Heywood, and A. N. Zincir-Heywood. 2016. On the Impact of Class Imbalance in GP Streaming Classification with Label Budgets. In *European Genetic Programming Conference*. 35–50.
[14] S. Khanchi, M. I. Heywood, and A. N. Zincir-Heywood. 2017. Properties of a GP active learning framework for streaming data with class imbalance. In *ACM Genetic and Evolutionary Computation Conference*. 945–952.
[15] K. Krawiec and M. I. Heywood. 2017. Solving Complex Problems with Coevolutionary Algorithms. In *ACM Genetic and Evolutionary Computation Conference (Companion)*. 782–806.
[16] P. Lichodzijewski and M. I. Heywood. 2008. Managing team-based problem solving with symbiotic bid-based genetic programming. In *ACM Genetic and Evolutionary Computation Conference*. 363–370.
[17] P. Parveen, J. Evans, B. M. Thuraisingham, K. W. Hamlen, and L. Khan. 2011. Insider Threat Detection Using Stream Mining and Graph Mining. In *IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*. 1102–1110.
[18] P. Parveen and B. M. Thuraisingham. 2012. Unsupervised incremental sequence learning for insider threat detection. In *IEEE International Conference on Intelligence and Security Informatics*. 141–143.
[19] T. Rashid, I. Agrafiotis, and J. R. C. Nurse. 2016. A New Take on Detecting Insider Threats: Exploring the Use of Hidden Markov Models. In *ACM CCS International Workshop on Managing Insider Security Threats*. 47–56.
[20] S. Ren, Y. Lian, and X. Zou. 2014. Incremental Naïve Bayesian Learning Algorithm based on Classification Contribution Degree. *Journal of Computers* 9, 8 (2014), 1967–1974.
[21] T. E. Senator, H. G. Goldberg, A. Memory, W. T. Young, B. Rees, R. Pierce, D. Huang, M. Reardon, D. A. Bader, E. Chow, I. A. Essa, J. Jones, V. Bettadapura, D. H. Chau, O. Green, O. Kaya, A. Zakrzewska, E. Briscoe, R. L. Mappus IV, R. McColl, L. Weiss, T. G. Dietterich, A. Fern, W.-K. Wong, S. Das, A. Emmott, J. Irvine, J. Yoon Lee, D. Koutra, C. Faloutsos, D. D. Corkill, L. Friedland, A. Gentzel, and D. D. Jensen. 2013. Detecting insider threats in a real corporate database of computer usage activity. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1393–1401.
[22] W. T. Strayer, D. E. Lapsley, R. Walsh, and C. Livadas. 2008. Botnet Detection Based on Network Behavior. In *Botnet Detection: Countering the Largest Security Threat*. 1–24.
[23] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson. 2017. Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams. In *Proceedings of the AAAI-17 Workshop on Artificial Intelligence for Cyber Security*. 224–231.
[24] A. Vahdat, J. Morgan, A. R. McIntyre, M. I. Heywood, and A. N. Zincir-Heywood. 2015. Evolving GP Classifiers for Streaming Data Tasks with Concept Change and Label Budgets: A Benchmarking Study. In *Handbook of Genetic Programming Applications*. 451–480.
[25] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia II, X. Xing, Liu X, and C. L. Giles. 2017. Adversary Resistant Deep Neural Networks with an Application to Malware Detection. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1145–1153.
[26] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. F. M. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. 2008. Top 10 algorithms in data mining. *Knowledge Information Systems* 14, 1 (2008), 1–37.
[27] I. Zliobaite, A. Bifet, B. Pfahringer, and G. Holmes. 2014. Active Learning With Drifting Streaming Data. *IEEE Transactions on Neural Networks Learning Systems* 25, 1 (2014), 27–39.