Approaches to Online Learning and Concept Drift for User Identification in Computer Security

Terran Lane and Carla E. Brodley

School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285
terran,brodley@ecn.purdue.edu

Abstract

The task in the computer security domain of anomaly detection is to characterize the behaviors of a computer user (the `valid', or `normal' user) so that unusual occurrences can be detected by comparison of the current input stream to the valid user's profile. This task requires an online learning system that can respond to concept drift and handle discrete non-metric time sequence data. We present an architecture for online learning in the anomaly detection domain and address the issues of incremental updating of system parameters and instance selection. We demonstrate a method for measuring direction and magnitude of concept drift in the classification space and present approaches to the above stated issues which make use of the drift measurement. An empirical evaluation demonstrates the relative strengths and weaknesses of these techniques in comparison to a number of baseline techniques. We show that, for some users, our drift adaptive techniques are advantageous.

Introduction

In this paper we examine methods for learning to classify temporal sequences of nominal data as similar to or different from previously observed sequence data when the underlying concept is subject to drift. This problem arises from the computer security task of *anomaly detection* (Kumar, 1995). The goal in this domain is to characterize the behaviors of a computer user (the `valid', or `normal' user) with a profile so that unusual occurrences can be detected by comparing a current input stream to the profile. This task presents us with a number of challenging machine learning issues including learning from discrete, non-metric time sequence data, learning from examples from only a single class, online learning, and learning in the presence of concept drift.

The goal of the anomaly detection domain is to produce an agent which can detect, through observations of system state, audit logs, or user generated events, when a user or system deviates from `normal' behavior. The presumption is that malicious behavior, especially on the part of an intruder who has penetrated a system account, will appear different from normal behavior in terms of some function of the present and historical observations of system state (Anderson, 1980; Denning, 1987). In this paper we refer to the individual observations as *events*. Taken over time, the events form an unbroken stream of temporally distributed nominal data. Our work focuses on an anomaly detection agent as a personal assistant that aids a single user in protecting his or her account from abuse. The alternative approach, of characterizing the *system's* state as normal or anomalous, entails a somewhat different set of problems and is examinined in, for example, (Lunt, 1990; Forrest, Hofmeyr, Somayaji & Longstaff, 1996; Lee, Stolfo & Chan, 1997). The learning task for our domain is to form a *profile* describing the valid user's normal patterns of behavior, and to use that profile to classify incoming events as belonging to or differing from the valid user. This task is made yet more difficult because the definition of what constitutes `normal' behavior for a particular user is subject to change over time as tasks, knowledge, and skills change. We envision the techniques presented here as working in conjunction with other methods such as biometric measurements and attack signature detection to create an overall accurate and robust security assistant.

Because the space of possible malicious behaviors and intruder actions is potentially infinite, it is impractical to characterize normal behavior as a contrast to known abnormal behaviors (Spafford, 1998). It is also desirable, for privacy reasons, that an anomaly detection agent only employ data that originates with the profiled user or is publicly available — an important criterion to much of the computer security community. This requirement leads to a learning situation in which only instances of a single class (`valid user') are available.

In this environment, the anomaly detection agent sees only an unbroken and undifferentiated stream of incoming events and must classify each event as anomalous or normal. The associated learning task (training the agent to recognize a particular user) possesses a number of difficulties not faced by traditional, static learning tasks. In particular:

Concept drift: A user's behaviors and tasks change with time. The anomaly detection agent must be capable of adapting to these changes while still recognizing hostile actions and *not* adapting to those.

Online learning: There is no division of `training data' versus `testing data'. Instead, the agent is presented with a homogeneous instance stream and must select appropriate

training instances itself.

Single class learning: The agent is only provided with examples from a single class (the normal user's data) for learning.

Temporal sequence learning: Many learning algorithms process instances composed of attributes and classes defined on a fixed attribute space. This representation is not particularly amenable to the unbroken stream data available in this domain. Either a mapping from the one to the other must be sought, or an algorithm designed for the native temporal data space must be found.

In other work, (Lane & Brodley, 1997a), we have explored some of the data representation and single class learning issues associated with the anomaly detection domain. The purpose of this paper is to explore issues associated with online learning and concept drift.

Structure of the Learning Component

Previously we have examined a static model of learning for anomaly detection in which separate train, parameter selection, and test data sets are available (Lane & Brodley, 1997a; Lane & Brodley, 1997b; Lane & Brodley, 1997c). An online learning system does not have the luxury of such distinctions. Incremental methods are needed to select instances for insertion into the model and to update current model parameters. In this section, we describe the structure of the learning component of the anomaly detector and enumerate the methods employed in our online system.

Token and Similarity Streams The incoming stream of tokens (events) is segmented into overlapping fixed-length sequences. The choice of the sequence length, l, was explored in (Lane & Brodley, 1997a) where it was found to depend on the profiled user. While not optimal for all users, the value $l = 10$ was found to be an acceptable compromise among the users tested there. Each sequence is then treated as an instance in an l-dimensional space and is compared to the known profile. The profile is a set, $\{T\}$, of previously stored instances and comparison is performed between all $y \in \{T\}$ and the test sequence via a similarity measure. Similarity is defined by a measure, $Sim(x, y)$, which makes a point-by-point comparison of two sequences, x and y, counting matches and assigning greater weight to adjacent matches. Similarity to the profile $Sim_{\{T\}}(x)$, is defined by: $Sim_{\{T\}}(x) = \max_{y \in \{T\}} Sim(x, y)$. This is the 1-nearest-neighbor rule of IBL on the non-Euclidean space defined by Sim. This measure, and some alternatives, is described and evaluated in (Lane & Brodley, 1997c).

Comparison of successive incoming sequences yields a similarity stream representing the similarity over time of the observed user to the profiled user. This signal turns out to be quite noisy, so it is smoothed with a trailing window mean value filter with window length w. Because classification takes place after smoothing, the window length limits the shortest time in which initial detection of an intruder can be made. We choose w to be 80 sequences — the minimum window length that we have found to reduce noise acceptably. It has been argued that an intruder can do a great deal of damage in less than $l + w = 90$ tokens and that alternative approaches should be explored for this reason. While the danger of short-term attacks is undeniable, there are also large classes of attackers who exploit a system over longer time periods (see (Stoll, 1989) for one example) and we orient our detector toward such attacks. Furthermore, an alternate branch of computer security research focuses on pattern matching detectors for locating known short-time attack signatures (for example, (Kumar, 1995)).

Classification Classification is performed on each point of the smoothed similarity stream, yielding a value of 0 (anomalous) or 1 (normal) for each time step. Classification is subject to two types of errors: false acceptance (incorrectly identifying hostile behaviors as normal) and false alarm (incorrectly flagging normal behaviors). It's important that the false alarm rate be low for the system to be usable, but we wish to induce as little false acceptance as possible for that cost. Because only a single class is available for training (the valid user), it's not possible to construct a Bayes-optimal classifier. The classification rule we employ, therefore, is:

$$class(x) = \begin{cases} 1 & \text{if} \quad P_{\{T\}}(x) \geq r \\ 0 & \text{if} \quad P_{\{T\}}(x) < r \end{cases}$$

for `acceptable' false alarm rate, r, where $P_{\{T\}}(x)$ denotes the probability of observing similarity value x given user profile $\{T\}$. As it turns out, all $P_{\{T\}}$'s that we have observed are characterized by a single strong peak with low-probability noisy tails. So, in this case, the above-stated classification rule can be approximated as:

$$class(x) = \begin{cases} 1 & \text{if} \quad t_{min} \leq x \leq t_{max} \\ 0 & \text{else} \end{cases}$$

where t_{min} and t_{max} are classification thresholds in the similarity measure space.

System Initialization The question of initializing an anomaly detector to a user's behavioral patterns in a *secure* fashion is a complex one and is beyond the scope of this paper. For the work described here, we *assume* that an adequate sample (a thousand tokens, in this paper) of intruder-free data is available for the profiled user. In (Lane & Brodley, 1997b), we show empirically that a thousand tokens is often sufficient to characterize a large, but not complete, segment of user behaviors. During system initialization all sequences are automatically classified as valid and incorporated into the profile. Sequences *are* compared to the extant profile before insertion, however, to accumulate usage and similarity statistics for use by parameter and instance selection methods (see below).

To set initial classification thresholds, the system compares incoming sequences to the current profile to accumulate a similarity value frequency histogram that approximates $P_{\{T\}}(x)$. With this distribution and an `acceptable' false alarm rate, r, we can calculate the decision boundaries, t_{max} and t_{min} such that the *a posteriori* probability outside the thresholds is r. For this paper, r was chosen to be 2%.

Instance Selection For each classified point after initialization, the learning system needs to decide whether to add the point to the profile or to reject it. This decision is especially critical in the presence of concept drift, when an unknown behavior pattern might represent the valid user changing tasks or might represent an intruder. A common approach to dealing with concept drift is to incorporate instances misclassified by the current model (Utgoff, 1989; Aha & Kibler, 1989). Such an approach is not appropriate to the anomaly detection domain because a training signal is not available to the learner to inform it that it has made a misclassification. Furthermore, storing instances about which the learner is extremely uncertain (i.e. have low similarity to the profile) as is done in (Lewis & Catlett, 1994), has the potential danger of assimilating hostile actions into the profile. Sequences labeled as abnormal are, therefore, not included in the profile. For the sequences labeled normal, we have examined four storage heuristics. The *keep* heuristic simply preserves all normal sequences. The converse policy, *reject*, refuses all sequences. An intermediate policy, *uncertain*, attempts to focus on sequences about which the profile is uncertain yet still labels normal. Under this heuristic, a sequence is assigned a probability of insertion as follows: if $t_{min} \leq Sim'_{\{T\}}(x) \leq t_{max}$ then,

$$P_{ins}(x) = k\frac{t_{max} - Sim'_{\{T\}}(x)}{t_{max} - t_{min}}$$

otherwise, $P_{ins}(x) = 0$, where $Sim'_{\{T\}}(x)$ denotes the smoothed similarity value of sequence x with respect to profile $\{T\}$, and k is a constant selected to make P a probability distribution. The final instance selection heuristic is *DAIP* (Drift Analysis Insertion Policy) which selects sequences for insertion only when a measure of concept drift indicates that the profile needs to be updated. Measurement of drift and the full description of this heuristic are deferred to the next section.

Parameter Updating Learning parameters such as sequence length, smoothing window length, and classification thresholds are all, potentially, dynamically adjustable. We focus here on t_{max} and t_{min} — the classification thresholds. After initialization, there are three methods available for updating the classification boundaries. *Entire* recalculates the thresholds at every time step from the similarity histogram of the entire profile at that time. *Windowed* calculates the thresholds only from a window of points within the profile. For this work, we take the window to be the same size as the initial profile — 1000 tokens — and to be drawn from the most recently acquired sequences. Finally, *DATA* (Drift Analysis Threshold Adjustment), adjusts the extant thresholds via a measure of the concept drift. A complete description of DATA is deferred to the next section.

Measurement of Drift

Concept drift can take place at many time scales, varying from a few tokens (perhaps stopping to read email in the middle of writing a conference article) to change over many months (changing research focus over time, for example). At the shortest time scales, drift phenomena are difficult to distinguish from noise. We focus on drift occuring at longer time scales — weeks to months — appearing as changes in the stream of similarity-to-profile measurements.

To quantify one class of drift effects, we calculate the best fit line (in a mean squared error sense) over a window of the similarity signal. The window size is chosen to be long enough to suppress most of the noise effects and yet short enough to be responsive to the scales of interest. We have found empirically that 1000 tokens is an acceptable length, but are currently examining methods for selecting this size automatically. The window size defines the scale at which we are measuring drift. Because we have no wish to adapt to behaviors `known' to be hostile, we calculate the best fit line only over instances classified (by the current model) as normal. The coefficient of the linear term of the best fit line then gives us an indication of the general directional trend of the similarity stream, and we take this value to be our drift measure, $\widehat{\Delta}$.

We employ our drift measure in two learning models (as discussed in Section). In the `DAIP` (Drift Analysis Insertion Policy) model, we employ the sign of $\widehat{\Delta}$ for instance selection. When $\widehat{\Delta} \geq 0$, the similarity measure is generally stable or increasing and the profile is doing a good job of matching current user behaviors. To prevent the profile size from increasing without bound, we do not insert sequences that are already covered by the current profile. When $\widehat{\Delta} < 0$, then the profile is performing poorly, and so is updated by inserting new instances.

The `DATA` (Drift Analysis Threshold Adjustment) model employs both the sign and magnitude of $\widehat{\Delta}$ for parameter estimation. `DATA` begins with the classification thresholds, $t_{max}(0)$ and $t_{min}(0)$ selected during system initialization, and updates them at each time step by adding $\widehat{\Delta}$: $t_{\{\text{max,min}\}}(i+1) = t_{\{\text{max,min}\}}(i) + \widehat{\Delta}(i)$. Under this model, the `width' or discrimination of the thresholds $(t_{max} - t_{min})$ remains unchanged for the lifetime of the system.

Empirical Evaluation

In this section, we describe learning models, data sources, and experimental structure and give results evaluating the learning models previously described.

Models examined For time and space reasons, we have not tested all of the twelve possible combinations of instance selection and parameter updating policies. Instead, we have focused on examining each issue (parameter updating and instance selection) separately. We have constructed six classifiers based on the previously described heuristics. The learning models we have examined are summarized in Table 1.

While some of the names we assign to learning models have obvious meanings (`truncate` and `random`), others bear some explanation. `P-opt` is `pseudo-optimal'. This model retains *all* valid instances of a user's behavior and is

Model	Select	Update
P-opt	keep	entire
truncate	reject	entire
W-opt	keep	window
DAIP	DAIP	entire
DATA	keep	DATA
U-ins	uncertain	entire

Table 1: Learning models as combinations of selection and update heuristics.

capable of finding the best possible similarity measure for a new sequence given prior experience. As we will see, however, making the best similarity match does not necessarily equate to having the best overall performance. Similarly, W-opt preserves all instances for the similarity calculation, but selects thresholds only over a window. Finally, U-ins selects instances for inclusion in the profile based on their uncertainty (i.e. proximity to the minimum acceptable similarity threshold).

Data Sources and Structure Of the thousands of possible data sources and features that might characterize a system or user, we chose to examine UNIX shell command data. We did so for two primary reasons: first, our interest is mainly in methods of characterizing human behavioral patterns and command traces reflect this more directly than do, say, CPU load averages and, second, shell data is simple and convenient to collect.[1] Lacking shell traces of actual intrusive or misuse behaviors, we demonstrate the behavior of the system on traces of normal system usage by different users. In this framework, an anomalous situation is simulated by testing one user's command data against another user's profile. This represents only a subset of the possible misuse scenarios — that of a naive intruder gaining access to an unauthorized account — but it allows us to evaluate the approach.

We have acquired shell command data from eight different users over the course of more than a year. The data events were tokenized into an internal format usable by the anomaly detector. In this phase, command names and behavioral switches were preserved, but file names were omitted under the assumption that behavioral patterns are at least approximately invariant across file names. The pattern `vi <file> gcc <file> a.out', for example, represents the same class of action regardless of whether file is homework1.c or cluster.c.

Baselining the system We are interested in baselining our techniques against currently implemented anomaly detection systems but it turns out to be difficult to do so, as we have not encountered published accuracies for other anomaly detection systems. In fact, according to Spafford (1998), with the exception of IDIOT (Kumar, 1995), performance measures for intrusion and anomaly detection systems have not been released in a refereed publication.[2] The exceptions to this general state are systems with roots in the machine learning community such as (Forrest, Hofmeyr, Somayaji & Longstaff, 1996) or (Lee, Stolfo & Chan, 1997). These systems concentrate on examination of data from privileged system tasks and are oriented toward characterization of programs and systems rather than users — a related but distinct task. Lacking baseline performance data, we have begun a project to implement some of the user-based anomaly detection algorithms described in the security literature.

It will also be noted that the false alarm rate for many of the techniques displayed below are relatively high. In fact, these rates *are* unacceptably high for a *standalone* anomaly detection system. We envision the techniques proposed here, however, as only a segment of a larger anomaly detection system, working in tandem with other detectors (possibly in a meta-learning framework).

Adaptation to Drift Concept drift in the anomaly detection domain can only occur between the valid user's past and present behaviors. Changes in the observed patterns of usage attributable to another user are not drift but anomalies. Thus, we are interested in measuring two quantities for a learning model: the true acceptance rate over time (representing the ability of the model to adapt to drift) and the true detection rate independent of time (representing the ability of the model to differentiate anomalies from drift). To measure these quantities, we constructed 42 simulated `attack' traces for each user. For each user we began by building six `base' traces of lengths one, two, five, ten, and fifteen thousand tokens drawn from that user's data. Each base trace was then converted into seven final traces by appending a block of one thousand tokens from each other user's data. The true acceptance rate is then the accuracy of a model on the basal part of a data trace, while the true detection rate is the accuracy on the final thousand tokens of a trace. Examination of true acceptance across the basal traces yields a view of acceptance over time. Similarly, examination of true detection rate across the five time steps gives us an indication of a model's ability to preserve correct detections while attempting to model drift.

We measured the performance of each of the six learning models described above. We are examining two axes simultaneously in these experiments: parameter measurement and instance selection policies. We present results for each class of learning model in turn below. Merely presenting averages summaries of each technique's performance over all data sets does not not reveal the true structure of the space, because such summaries have high variances for this domain. Instead, we present extreme and characteristic case

[1] The techniques discussed here could, of course, be extended to cover any discrete stream of nominal values such as system call logs, keystrokes, or GUI events. Furthermore, this classifier could likely be combined with classifiers based on other measures to yield a system with higher overall performance.

[2] IDIOT is an intrusion detection system which employs a pattern matching algorithm to detect known attack signatures in audit data. Its patterns are not intended to generalize to unknown cases, so rather than accuracy, time and space performance measures are reported.

Model	Test User	Elapsed time (thousands of tokens) 1	2	5	10	15
True accept rate (%)						
P-opt	S	100.0	97.8	89.2	81.0	80.9
W-opt	S	100.0	97.1	82.2	55.9	51.3
DATA	S	100.0	97.9	89.6	83.3	84.2
True detect rate (%)						
P-opt	U2	5.4	5.5	5.7	6.8	6.9
	U4	52.2	52.8	54.2	54.7	55.1
	U6	16.2	17.3	18.4	26.2	21.5
W-opt	U2	7.9	10.4	68.5	100.0	96.3
	U4	52.0	58.8	73.9	100.0	79.4
	U6	16.2	96.7	95.7	100.0	97.7
DATA	U2	5.6	5.6	8.9	11.9	17.0
	U4	52.6	53.0	59.7	62.5	66.8
	U6	20.1	27.2	40.2	94.4	95.3

Table 2: Results for parameter selection models on USER1's profile.

Model	Test User	Elapsed time (thousands of tokens) 1	2	5	10	15
True accept rate (%)						
DAIP	S	100.0	93.9	73.1	61.4	57.9
trunc	S	100.0	94.8	81.4	78.7	80.8
U-ins	S	100.0	92.2	75.2	65.7	62.9
True detect rate (%)						
DAIP	U0	0.0	6.6	10.8	23.6	31.2
	U1	18.2	25.4	30.2	43.6	41.3
	U4	57.9	66.5	65.2	64.2	73.1
trunc	U0	5.1	5.1	5.1	11.3	5.1
	U1	11.3	11.3	11.3	14.9	11.3
	U4	50.1	50.1	50.1	55.0	50.1
U-ins	U0	0.0	1.5	1.2	12.2	16.5
	U1	14.2	16.5	19.5	31.1	31.7
	U4	62.3	65.2	68.3	69.2	67.5

Table 3: Results for instance selection models on USER5's profile.

behaviors for the various models to indicate the strengths and weaknesses of each approach.

Parameter selection methods Table 2 displays relative behaviors for the three tested parameter adaptation methods. Recall that all of these methods use the `keep' instance selection strategy. We find, here, that P-opt has strong true accept performance but weak true detection rates. Recall that P-opt sets its classification thresholds based on its entire previous experience. While this allows it to recognize a broad range of behaviors, it has the twin difficulties that the decision boundaries become widely spaced (thus increasing false acceptance) and that it becomes difficult to adjust the decision boundaries quickly in response to changing circumstance. As an attempt to minimize the second problem, we introduced the W-opt model. This learning strategy also preserves all known valid instances for the purpose of similarity measurement, but selects classification boundaries based only on data from a window of those instances. While W-opt has substantially superior true detect rates, it suffers in true accept rates.[3] This model is setting tighter decision boundaries than is the P-opt model and can adapt more quickly, but is unable to predict changing usage patterns. Analysis of similarity value frequency over the trailing window only gives W-opt an idea of where the concept *was*, not where it is *going to be*. A useful balance is struck between the two extremes by DATA. This model begins with the classification boundaries selected during system initialization and updates them in response to the large scale measure of drift described above. Again, the update is based only on data from a recent window of experience, but DATA has some indication of where the similarity value concept will be in the immediate future. Finally, DATA prevents the decision region from becoming too narrow by preserving the initially selected width (i.e. $t_{max} - t_{min}$ is constant).

Overall, we find that DATA matches or outperforms P-opt approximately 70% of the time on both true accept and true detect. Conversely, W-opt beats DATA on 68% of the true detect cases, but loses to it in 81% of the true accept tests.

Instance selection methods One class of behaviors for instance selection models is displayed in Table 3. These models employ the `entire' parameter selection method. The truncate model is, in this case, equivalent to merely employing the static classifier trained during system initialization. Its true detect rate is effectively constant; the variations observed here are effects of the smoothing filter which can `carry over' high similarity values from the valid user's data into the hostile user's data. As the `valid user' concept drifts, this static model cannot adapt and true accept accuracy drops. The `intelligent' instance selection methods experience a more drastic drop in this case, in exchange for increasing detection accuracy. The problem here seems to be less in the particular method of instance selection, but in the fact of instance selection itself. All models discussed in this paper accumulate only instances that are `known' (by the current model) to be valid. If the current model does not encompass a particular change in concept, then all behaviors associated with that change are lost, and the decision boundaries become correspondingly narrower. As truncate is a static model, it is not subject to such loss. Though DAIP makes an effort to account for changing concept, it appears to fail for this user. U-ins performs better (in both accept and detect rates), apparently because it focuses explicitly on uncertain instances and, thus, accepts a wider class of behaviors than does DAIP.

The converse situation is displayed in Table 4. Here DAIP and U-ins are quite effective at identifying the true user,

[3] Given a tradeoff, it is generally preferable in this domain to have strong accept rates rather than strong detect rates. High false alarm rate renders the security system annoying or unusable, while high false accept rates only delay detection. This may allow an intruder to do more damage but, in the end, the intruder need only be caught once.

Model	Test User	Elapsed time (thousands of tokens) 1	2	5	10	15
True accept rate (%)						
DAIP	S	100.0	91.5	92.9	95.3	96.5
trunc	S	100.0	83.7	81.8	82.9	82.9
U-ins	S	100.0	100.0	92.2	94.4	94.9
True detect rate (%)						
DAIP	U3	95.0	81.0	93.7	69.8	52.9
	U4	95.0	24.4	38.6	26.7	26.2
	U6	94.7	41.2	50.0	37.6	34.9
trunc	U3	96.0	96.5	94.6	97.6	96.1
	U4	96.3	96.5	94.8	97.4	96.1
	U6	95.7	95.6	93.5	95.6	95.2
U-ins	U3	83.5	82.0	85.3	74.5	71.0
	U4	13.8	21.8	27.8	33.2	34.2
	U6	94.6	41.4	75.6	43.8	50.6

Table 4: Results for instance selection models on USER2's profile.

but are far more lax in detecting hostile actions. In this case, truncate's static classifier turns out to be more accurate at discriminating foreign behaviors. Now the narrow concentration of the adaptive methods serves them, as this user's behaviors seem concentrated to a narrower class and to experince less drift than the behaviors of other users. (Manual examination of the history traces verify this observation.) Because the static decision boundaries were not originally selected optimally, truncate is restricted to a more-or-less constant false alarm rate, while the other methods are free to adapt to more accurate hypotheses. The tradeoff is that there appears to be a certain degree of overlap between USER2's behaviors and those of the other users. The adaptive methods seem to focus the profile and decision thresholds into this layer — as hostile behaviors are added to the profile it becomes progressively easier to add more hostile behaviors and the false accept error rate grows quickly.

Overall, we find that DAIP matches or outperforms truncate in 52% of the true accept tests, but loses to it on true detect nearly 85% of the time. Interestingly, the cases in which DAIP wins are concentrated into a few profiles for true accept tests and a few profile/attacker pairs for true detect tests. This indicates that there may be many sorts of drift taking place, and that the DAIP bias is appropriate to only some of them. The disparity is even greater for U-ins who beats truncate 60% of the time on true accept but loses 90% of the time on true detect tests.

Conclusions

We have examined some of the difficulties involved in tracking user behaviors over time for use in the computer security task of anomaly detection. We demonstrated an online learning system for this domain, and described some of the issues inherent in incremental learning with no training signal. In particular, we investigated techniques for updating hypothesis parameters and selecting instances for insertion into the user profile. We found that, although there is high variability in the strengths and weaknesses of the various techniques, intelligent methods exist for each of these areas. A measure of drift based on estimating its magnitude and direction in a continuous, 1-D feature space was found to be useful (in up to 70% of the cases) both for updating parameters and for selecting instances for inclusion in the profile. An instance selection method based on uncertainty sampling was also found to have areas of strength.

The high variability in regions of effectiveness of the various techniques suggests two possible directions for future work in this area. First, we hope to be able to exploit complementary strengths in different learning models through the use of hybrid systems. In this paper we investigated each phase of the overall learning model (parameter selection and instance selection) separately. Our hope is that intelligent combination of techniques from each can lead to a stronger overall system. The second avenue of exploration is to attempt to exploit overlapping strengths through a form of meta-learning. We have observed that some techniques (truncate and DATA, for example) yield generally uncorrelated results, making them tempting models for use with meta-learning.

Finally, we are interested in examining more sophisticated measurements of concept drift. The measurement used in this paper only tracks the general direction and amount of change of the concept of interest. We would also like to be able to track and predict the complete envelope of the classification region, as well as other system parameters such as window lengths and profile size.

In conclusion, we have presented methods for an online learning system for anomaly detection. Although error rates are too high to be of use for a standalone system, in combination with other user classification techniques such as biometric measurements or model-based behavioral analysis, these techniques may form a valuable part of a robust security system.

References

Aha, D. W., & Kibler, D. (1989). Noise-tolerant instance-based learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 794-799). Detroit, Michigan: Morgan Kaufmann.

Anderson, J. P. (1980). *Computer security threat monitoring and surveillance*, (Technical Report), Washington, PA, James P. Anderson Co.

Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering, 13*, 222-232.

Forrest, S., Hofmeyr, S. A., Somayaji, A., & Longstaff, T. A. (1996). A sense of self for Unix processes. *Proceedings of 1996 IEEE Symposium on Computer Security and Privacy.*

Kumar, S. (1995). *Classification and detection of computer intrusions.* Doctoral dissertation, Department of Computer Sciences, Purdue University, W. Lafayette, IN.

Lane, T., & Brodley, C. E. (1997a). *Detecting the abnormal: Machine learning in computer security*, (TR-ECE 97-1), West Lafayette, IN: Purdue University.

Lane, T., & Brodley, C. E. (1997b). An application of machine learning to anomaly detection. *National Information Systems Security Conference.*

Lane, T., & Brodley, C. E. (1997c). Sequence matching and learning in anomaly detection for computer security. *Proceedings of AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management.*

Lee, W., Stolfo, S., & Chan, P. (1997). Learning patterns from UNIX process execution traces for intrusion detection. *Proceedings of AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management.*

Lewis, D., & Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. *Machine Learning: Proceedings of the Eleventh International Conference* (pp. 148-156). New Brunswick, NJ: Morgan Kaufmann.

Lunt, T. F. (1990). IDES: An intelligent system for detecting intruders. *Proceedings of the Symposium: Computer Security, Threat and Countermeasures.* Rome, Italy.

Spafford, E. H. (1998). *Personal communication.*

Stoll, C. (1989). *The Cuckoo's Egg.* Pocket Books.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning, 4,* 161-186.