Machine Learning, 51, 73–107, 2003

An Empirical Study of Two Approaches to Sequence Learning for Anomaly Detection

TERRAN LANE terran@cs.unm.edu
Department of Computer Science, University of New Mexico, Albuquerque, NM, USA

CARLA E. BRODLEY brodley@ecn.purdue.edu
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA

Editor: David W. Aha

Abstract. This paper introduces the computer security domain of anomaly detection and formulates it as a machine learning task on temporal sequence data. In this domain, the goal is to develop a model or *profile* of the normal working state of a system user and to detect anomalous conditions as long-term deviations from the expected behavior patterns. We introduce two approaches to this problem: one employing instance-based learning (IBL) and the other using hidden Markov models (HMMs). Though not suitable for a comprehensive security solution, both approaches achieve anomaly identification performance sufficient for a low-level "focus of attention" detector in a multitier security system. Further, we evaluate model scaling techniques for the two approaches: two clustering techniques for the IBL approach and variation of the number of hidden states for the HMM approach. We find that over both model classes and a wide range of model scales, there is no significant difference in performance at recognizing the profiled user. We take this invariance as evidence that, in this security domain, limited memory models (e.g., fixed-length instances or low-order Markov models) can learn only part of the user identity information in which we're interested and that substantially different models will be necessary if dramatic improvements in user-based anomaly detection are to be achieved.

Keywords: anomaly detection, application, instance-based learning, hidden Markov models, computer security

1. Introduction

Automated modeling of human behaviors is useful in the computer security domain of anomaly detection (Anderson, 1980; Denning, 1987). In this domain, the task is to develop a model or *profile* of the normal working state of a computer system, network, or user with the ultimate goal of detecting anomalous conditions as deviations from expected behavior patterns. In this article, we examine the problem of modeling users' normal work patterns and detecting abnormalities relative to their profiles. This definition of anomaly detection encompasses not only *intrusions* by external agents but also *insider abuses* by authorized site users (possibly including the account owner). In practice, many modern intrusion detection systems are hierarchical and distributed and include dozens or hundreds of low-level sensory agents, of which a user-level anomaly detection sensor of the type we describe may be only one (Balasubramaniyan et al., 1998; Porras & Neumann, 1997; Cis, 1999; ISS, 2000).

The anomaly detection problem can be framed as one of learning a binary concept (valid user versus anomaly) on a domain consisting of a temporal sequence of discrete, unordered elements. For example, the data space may consist of command line strings, system call traces, network packet logs, or even GUI event streams. For convenience, we will consider time to be a discrete quantity and all observations to take place synchronously. The job of the classifier is to label each time step in the observation sequence as normal or abnormal. It may be insufficient to assign only a single label to an entire login session, as hostile behaviors may constitute only a subset of the session's data (as, for example, when a hostile co-worker preempts the console of someone who has negligently left the console unattended for some period).

In this work, we examine the task of learning user profiles at the user-interface level (as opposed to analysis of system call traces (Wespi, Darcier, & Debar, 1999; Forrest et al., 1996; Lee, Stolfo, & Chan, 1997), network packet traces (Lee, Stolfo, & Mok, 1998; Heberlein et al., 1990), or resource consumption patterns (Lunt, 1990)). In particular, we examine traces of UNIX command line (shell) data, though the techniques presented here can encompass any temporal sequence of discrete data. We consider a single user's shell history traces to form a single temporal stream of symbols (or tokens) drawn from a discrete alphabet. Tokens are whitespace separated words (command names, their arguments, and a count of the number of file names encountered on the command line) as well as shell metacharacters (such as pipes or redirections).

Modeling of humans for anomaly detection presents a number of challenging machine learning issues:

- Human-generated data are notoriously noisy.
- There are thousands of potential data sources to examine, many of which are defined on extremely large alphabets.
- Humans are non-stationary data sources.
- Because of privacy and completeness concerns, user profiling data are drawn from only a single user and represent single-class training data.
- The ratio of "normal" usage to "anomalous" or hostile activity is likely to be highly skewed.
- The tradeoff between error types (generating false alarms versus failing to identify impostors) is site-specific and depends on local security policy.
- The profile learner and classifier are presented with unbounded data streams from potentially very prolific data sources (e.g., system call traces).

For a more detailed analysis of these issues, please see Lane (2000).

In this article, we describe approaches to some of these problems based on two classes of learning techniques. The hidden Markov model (HMM) formulation of this task is a natural one as HMMs are explicit models of sequential data. In contrast, an instance-based learner (IBL) does not typically include an explicit model of sequentiality. Nevertheless, we demonstrate that, with a proper problem formulation, the IBL model yields comparable performance to the explicit time-series model of the HMM. We also examine techniques for learning temporal sequences of discrete data drawn from a large alphabet and address noise and model scaling issues. Our empirical results are generated according to a range of

misclassification costs, in a way compatible with Receiver Operating Characteristic (ROC) analysis for optimal classifier selection under varied cost and distribution assumptions (Provost & Fawcett, 1998).

2. Computer security goals

Before describing our anomaly detection sensor and empirical results in detail, we would like to give some attention to the scope of this work and the goals for this sensor. Though the ultimate goal of the anomaly detection domain is to produce a system capable of distinguishing all hazardous or intrusive anomalies from all normal behaviors, it is not necessary to fully achieve this target to make substantial contribution in this domain. Our goal is not to produce a complete, stand-alone anomaly detection system that would catch all impostors or to solve all security issues. Rather, we are operating within the context of two operational principles, one drawn from the computer security community and the other from the machine learning community:

Defense in depth: This principle is a result of the acknowledgement that in practice there is no perfect security. All security mechanisms are susceptible to some form of compromise, and substantially increasing the strength or reliability of any given mechanism may require disproportionate effort or cost (e.g., retinal scanners are very powerful biometric user identification systems but they require specialized hardware and software support that may be beyond the reach of many organizations). Thus, it is desirable to construct a layered network of imperfect but relatively cheap defenses. We can have high confidence in the resulting system because of the redundancy of defenses even if we have only moderate confidence in each individual mechanism (Pfleeger, 1997).

Hierarchical classification and data reduction: It is well known in the machine learning literature that appropriate combination of a number of weak classifiers can yield a highly accurate global classifier (Schaffer, 1994; Freund & Schapire, 1997). Although various forms of voting have been shown to be effective in many cases, a natural alternative in data-intensive domains is multi-layer or hierarchical classification. In such tiered decision systems, the function of the lowest layer of classifiers is to provide *focus of attention* (FOA)—that is, to reduce the enormous data inputs to a manageable load for the more computationally intensive upper layers of the hierarchy which are responsible for such tasks as feature identification and final classification. In practice, hierarchical classification systems have proven highly effective in such tasks as stellar classification (Fayyad, Weir, & Djorgovski, 1993), planetary geology (Burl et al., 1994; Stough & Brodley, 1997), and medical image analysis (Shyu et al., 1999). A number of recent projects in the computer security community have taken advantage of the power of hierarchical classification in the design of distributed intrusion detection systems (Balasubramaniyan et al., 1998; Porras & Neumann, 1997; Cis, 1999; ISS, 2000).

Thus, we do not aspire to provide a complete security solution but an incremental benefit. The sensor we describe in this article is intended to function as a focus of attention unit at the lowest level of a classification hierarchy; final decisions of hazard level and notification

of security officers is the responsibility of a higher level agent with access to distilled results from many data sources. Nor do we intend to consider all of the thousands of potentially relevant data sources; our sensor focuses only on human command-line level data, leaving analysis of, for example, system call traces or network loads, to other sensors. We are, in fact, currently engaged in testing our system as a sensory agent in the AAFID hierarchical intrusion detection system (Balasubramaniyan et al., 1998).

The objective, therefore, for this sensor is threefold:

- To provide a reasonable level of accuracy and data reduction to a higher level classifier in the decision hierarchy.
- To provide results in a timely manner so that security actions can be taken promptly.
- And to run efficiently, imposing little resource burden on the classification hierarchy and the computer system as a whole.

In terms of accuracy, we focus on ability to discriminate impostors, recalling that a false alarm on the valid user can potentially be discarded by a higher-level decision maker but that an impostor missed by this sensor will never be available to higher-level classifiers.

Finally, security incidents can take place at a wide range of time scales. We can roughly distinguish two classes of time scales: short-term attacks (consisting of a few tens of tokens at most), which are traditionally addressed via signature matching detectors (Kumar & Spafford, 1994; Cis, 1999; ISS, 2000; Gordon, 1996), and longer-term attacks, which are often approached with statistical or learning techniques (as in this article). Documented events in the latter class have occurred at such high-profile sites as the University of California at Berkeley, Mitre Corp. (Stoll, 1989), the Air Force's Rome Labs, Harvard University, and Citibank Corp. (Power, 1998).

3. Learning from nominal-valued temporal sequence data

Our data are tokens drawn from UNIX command line history traces, and are directly generated by humans. Humans produce a different class of data than do machine-level sources such as system call traces (Wespi, Darcier, & Debar, 1999; Forrest et al., 1996; Lee, Stolfo, & Chan, 1997) or network packet logs (Lee, Stolfo, & Mok, 1998; Heberlein et al., 1990). Machine-level data are almost exclusively produced by automated sources (sequences of system calls arranged by a compiler, for example) and display high repetition and little noise when compared to human-level data.

We assume the existence of some quantity of known "pure" data that is truly representative of the user's behaviors for the purposes of training the user model and selecting classification parameters. The problem of effectively and securely initializing such a system is a critical one for a fielded anomaly detection system, but we do not address that issue here. For some approaches to this problem see (Smaha, 1988; Heberlein et al., 1990).

In this work, we employ two different techniques for user modeling. In the first approach, using an instance-based learning (IBL: Aha, Kibler, & Albert, 1991) framework, the user model consists of a set of behavioral exemplars drawn from historically observed user data. The instances are fixed-length subsequences of the input data stream and instance

proximities are calculated via a domain-specific similarity measure. The nominal-valued temporal input sequence is mapped to a continuous-valued temporal sequence on which classification is performed via a similarity threshold test. In the second approach, the user model is a hidden Markov model (HMM) (Rabiner, 1989), which encodes a probability distribution over the space of discrete-valued temporal sequences. The likelihood of the input sequence is evaluated with respect to this model via the forward-backward algorithm, and classification is performed via a likelihood threshold test.

In this section we describe the training and operation of each approach. We discuss other methods for learning on temporal sequence data, and examine how the anomaly detection domain differs from other sequence learning domains.

3.1. Alternate approaches to sequence learning

In the anomaly detection domain the raw data consists of discrete, unordered (i.e., nominal-valued) elements such as command strings. For time series of numeric values, techniques such as spectral analysis (Oppenheim & Schafer, 1989), principle component analysis (Fukunaga, 1990), linear regression (Casella & Berger, 1990), linear predictive coding (Rabiner & Juang, 1993), nearest neighbor matching, (γ, ϵ)-similarity (Bollobás et al., 1997; Das, Gunopulos, & Mannila, 1997), and neural networks (Chenoweth & Obradovic, 1996) have proven fruitful. Such techniques typically employ a Euclidean distance or a related distance measure defined for real-valued vectors.

There are a number of learning algorithms that are amenable to learning on spaces with nominal-valued attributes. For example, decision trees (Quinlan, 1993) are well suited to representing decision boundaries on discrete spaces. The bias used to search for such structures generally (in axis-orthogonal decision trees) employs a greedy search that examines each feature independently of all others. This bias does not exploit internal relations arising from causal structures in the data generating process.

One method of circumventing this limitation is to convert the data to an atemporal representation in which the causal structures are represented explicitly. Norton (1994) and Salzberg (1995) each independently used such a technique for the domain of learning to recognize coding regions in DNA fragments. DNA coding, while not temporal, does exhibit interrelations between positions that are difficult for conventional learning systems to acquire directly. The features extracted from the DNA sequences were selected by domain experts, and cannot be generalized to other sequential domains. Although such an approach could be applied to the anomaly detection domain, it would require considerable effort on the part of a domain expert, and the developed features would apply only to that data source. We are interested in developing techniques that can be applied across different data sources and tasks.

There also exist learning methods explicitly developed to model sequence data. Methods for learning the structure of deterministic finite-state automata (DFA), for example, have been widely studied (Angulin, 1987; Rivest & Schapire, 1989; Aslam & Rivest, 1990). DFAs, however, are not well suited to modeling highly noisy domains such as human-generated computer interface data. If the data can be observed below the user interface level (i.e., after interpretation by the UNIX command shell), then many syntactic and semantic

errors will have been removed and the data will be cleaner. Yoshida and Motoda (1996) employ I/O relations at this level to develop finite-state graph models of user behaviors. The simplest extension of DFA models to noisy domains are Markov chain models (Davison & Hirsh, 1998), which allow stochastic state transitions. These models have the advantage that, unlike HMMs, the Maximum-Likelihood estimate for transition probabilities has a closed form. Markov chain models typically emit symbols deterministically (each state or arc emitting only a single symbol), requiring a state for each symbol of the alphabet, or $|\Sigma|^2$ total transition probabilities to be learned for an alphabet of size $|\Sigma|$. When the alphabet is large (in our empirical analyses, we have observed alphabets of over 2500 unique symbols), the dimensionality of the parameter space is high and the amount of training data required to accurately estimate low-probability transitions is very large. Finally, deterministic output Markov models with unique states (i.e., each symbol is emitted by only one state) can only represent a single context for any given symbol. In the anomaly detection domain symbols can have multiple contexts. The command `vi`, for example, can be employed for editing both source code and journal articles. An alternative formulation that represents multiple contexts with deterministic outputs are Markov trees (Laird & Saul, 1994). We intend to investigate such models in future work.

3.2. An IBL representation of anomaly detection

Although most instance-based methods do not explicitly model temporality or sequentiality, it is relatively straightforward to represent some classes of sequential relations within an IBL framework. In this section, we describe the use of an instance-based learner to model users' temporal-sequence behavioral data. We give a brief overview of the system's flow of information (shown schematically in figure 1) here and describe the individual components in more detail below. Data enters the system, in the upper left, as an undifferentiated sequence of discrete symbols (UNIX shell command lines in this work) and is passed through a parser

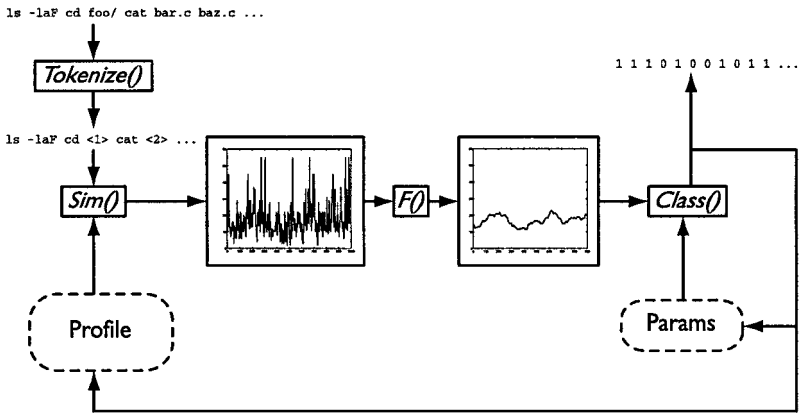

Figure 1. Information flow in the instance-based anomaly detection system.

(`Tokenize()`), which reduces the data stream to an internal format and does preliminary feature selection. The resulting data stream is compared to the user's historical profile via a similarity measure (`Sim()`), yielding a temporal sequence of real-valued similarity measures (indicating instantaneous similarity of observed data to the profile). Because the instantaneous similarity measure stream is highly noisy, classification based on this signal is difficult. To solve this problem, we introduce a noise-suppression filter (`F()`). Classification of the smoothed data stream is via a threshold decision module (`Class()`) whose decision boundaries are set through examination of an independent, parameter-selection set of the user's historical data. The final, binary class stream (upper right) is the detector's estimation of the current state of the input data (1 being "normal" and 0 "abnormal").

The components described above suffice to operate the detection system in a batch mode. In this mode, the detector accumulates a single, fixed profile and employs it for all further classifications. In an operational setting we face the additional difficulty that human behaviors are dynamic and what is considered "normal" behavior is likely to change over time. To respond to the presence of concept drift, an online learning method is required in which the detector employs the feedback loop shown in figure 1 to update the profile and classification thresholds. In the work described in this article, we are interested in examining the effects of profile data reduction techniques in isolation. To prevent interactions with the additional complexities of the online mode feedback loop, we perform all experiments in batch mode. Elsewhere, we have examined some of the online learning issues involved with adapting user models to changing behaviors (Lane & Brodley, 1998).

3.2.1. Feature extraction. In our environment we have been examining UNIX shell command data, captured via the `(t)csh` history file mechanism. The history data are parsed by a recognizer for the `(t)csh` command language, and are emitted as a sequence of tokens in an internal format. Each "word" of the history data (e.g., a command name, group of command flags, or shell meta-character) is considered to be a single token. The resulting alphabet is very large—over 35,000 distinct symbols in just the smaller of our two groups of users. Because the frequencies of some of these tokens are quite low, gathering adequate statistics over an alphabet this large is difficult. We have investigated different methods of reducing the alphabet size and have found that omitting file names in favor of a filename count (e.g., `cat foo.c bar.c gux.c` is converted to `cat <3>`) greatly constrains the alphabet size (to just over 2,500 distinct tokens) and improves accuracy. We have also examined the use of additional features such as file name, file extension, activity time-of-day, token class (e.g., command name, directory name, shell metacharacter), and working directory. Preliminary findings with these features are not promising, and it appears that more complex methods will be required to extract their utility.

3.2.2. The similarity measure. We have examined several measures for computing the similarity between two discrete-valued temporal instances (Lane & Brodley, 1997c). We describe here the measure that we have found, in empirical investigation, to perform best on average across users.

The similarity measure operates on token sequences of equal, fixed length. For a length, l, the similarity between sequences $X = (x_0, x_1, \ldots, x_{l-1})$ and $Y = (y_0, y_1, \ldots, y_{l-1})$ is

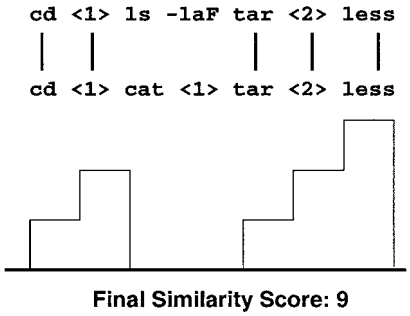

Figure 2. Example of sequence similarity calculation. Two sequences are compared, element by element. The bottom curve represents the weight contributed by each match, and the final similarity is the area under this curve.

defined by the pair of functions:

$$w(X, Y, i) = \begin{cases} 0 & \text{if } i < 0 \text{ or } x_i \neq y_i \\ 1 + w(X, Y, i-1) & \text{if } x_i = y_i \end{cases}$$

(where $w(X, Y, i) = 0$ for $i < 0$ so that $w(X, Y, 0)$ is well defined when $x_0 = y_0$) and

$$\text{Sim}(X, Y) = \sum_{i=0}^{l-1} w(X, Y, i).$$

The converse measure, *distance*, is defined to be:

$$\text{Dist}(X, Y) = \text{Sim}_{\max} - \text{Sim}(X, Y)$$

where $\text{Sim}_{\max}$ is the maximum attainable similarity value for a given sequence length: $\text{Sim}_{\max} = \text{Sim}(X, X)$.

An example similarity calculation is depicted in figure 2. The function $w(X, Y, i)$ accumulates weight linearly along matching subsequences (bottom curve), and $\text{Sim}(X, Y)$ is the integral of total weight over time (area under the weight curve). In the limiting case of identical sequences, this measure reduces to $\text{Sim}_{\max} = \sum_{i=1}^{l} i = \frac{l(l+1)}{2}$. Thus, a run of contiguous matching tokens will accumulate a large similarity, while changing a single token, especially in the middle of the run, can greatly reduce the overall similarity. This measure depends strongly on the interactions between adjacent tokens as well as comparisons between corresponding tokens in the two sequences (i.e., tokens at the same offset, i, within each sequence). The sequence length, l, is a user-dependent parameter and was explored in Lane and Brodley (1997a) where the best value was found to be dependent on the profile and the opponent being detected. Plausible values for l are small integers in the range $[8 \ldots 15]$ and the setting $l = 10$ was found to be an acceptable compromise across users.

A user profile is a collection of sequences, $\mathbf{D}$, selected from a user's observed actions. The similarity between the profile and a newly observed sequence, X, is defined to be:

$$\text{Sim}_{\mathbf{D}}(X) = \max_{Y \in \mathbf{D}} \{\text{Sim}(Y, X)\}.$$

This rule is related to the 1-nearest-neighbor classification rule (Fukunaga, 1990), although we are not performing classification at this stage but, rather, are defining similarity to known patterns. We have examined the possibility of using an average similarity to the entire profile, but found that such a measure had much lower accuracy than the measure given here. An average across the entire profile decreases the ability of the classifier to resolve local patterns in the classification space.

3.2.3. Segmenting the event stream. Because the similarity measure is defined only for fixed length sequences, it is necessary to partition the raw event stream into component subsequences. This raises the question of optimal sequence alignments: where should each sequence be defined to start? Our approach is based on the model scaling techniques presented in Section 5.1. The system initially segments the data stream into all possible overlapping sequences of length l (thereby replicating each token l times). Thus, every position, i, of the event stream is considered to be the starting point for a sequence of length l referred to as the ith sequence or the sequence at time step i. For example, under sequence length $l = 6$, the tokenized data stream "`cat <3> > <1> ls -l | more`" would be converted to the three sequences "`cat <3> > <1> ls -l`", "`<3> > <1> ls -l |`", and "`> <1> ls -l | more`".

3.2.4. Noise-suppression. In practice, we have found that the instantaneous similarity stream, produced by comparing an input data stream to a user profile, is far too noisy for effective classification. We attribute the high degree of noise to natural variations in the user's actions and patterns. For example, the user may temporarily suspend writing a paper to deal with urgent incoming email, thus disrupting his or her standard paper writing routine. Such a disruption will appear as a spuriously low similarity spike within an overall high similarity period. We therefore employ a noise reduction filter before selecting decision thresholds or performing classification. We employ a trailing window mean value filter defined as:

$$v_{\mathbf{D}}(j) = \frac{1}{W} \sum_{i=j-W+1}^{j} \mathrm{Sim}_{\mathbf{D}}(i)$$

where $\mathrm{Sim}_{\mathbf{D}}(i)$ is the similarity of the ith token sequence to the user profile $\mathbf{D}$, W is the window length, and $v_{\mathbf{D}}(j)$ is the final value of sequence j with respect to $\mathbf{D}$. In a comparison of the mean-value filter with a median-value filter, we found that, while the median filter is generally more effective at short window lengths ($W < 80$), performance for the two methods is approximately equivalent at longer window lengths (Lane & Brodley, 1997c). We use the mean-value filter here because we are employing $W = 100$ in this work and the mean filter can more easily be made to run quickly. While a great deal of damage can be inflicted in fewer tokens than than the window length, such short term attacks can be more readily handled by matching known attack signatures (Kumar & Spafford, 1994; Cis, 1999; ISS, 2000; Gordon, 1996). We are primarily concerned here with the class of long-term, low-profile attacks such as resource theft or industrial data theft. The time required to produce 100 tokens varies with user and task, but it can be as short as a few minutes during periods of intense work.

3.2.5. Classification and threshold selection. The similarity-to-profile measure defines a transformation from the original, l-ary nominal space to a one-dimensional, real-valued space in which a point set (command trace), $\mathbf{T}$, appears as a probability distribution, $P_{\mathbf{T}}$, over possible similarity values, $v \in 1 \ldots \frac{l(l+1)}{2}$. When multiple classes are observable their probability distributions can be used to construct Bayes-optimal decision boundaries (Fukunaga, 1990). In the anomaly detection domain, however, we possess data only from the profiled user; the Bayes-optimal boundary is unobservable to us. Furthermore, for most of the data sets we have examined, the *unweighted* Bayes-optimal threshold is overly critical of the profiled user. In figure 3, normal (U0) and anomalous (U1) similarity distributions are displayed together with the Bayes-optimal classification threshold and an alternative possible classification threshold (the acceptable false alarm threshold, described below). Sequences whose similarity to the profile falls to the right of the classification threshold are labeled normal while points falling to the left are labeled abnormal. The area under distribution U0 and to the left of the threshold is then the false alarm probability (the probability of the valid user being falsely accused of being anomalous) while the area under distribution U1 and to the right of the threshold is the probability of falsely accepting an anomalous user. In this example, employing the unweighted Bayes-optimal threshold for classification would yield an unacceptably high false alarm rate.

In light of the above, we must seek another method for selecting a decision boundary. Conveniently, the constraints of our domain provide us with a practical heuristic: constrain

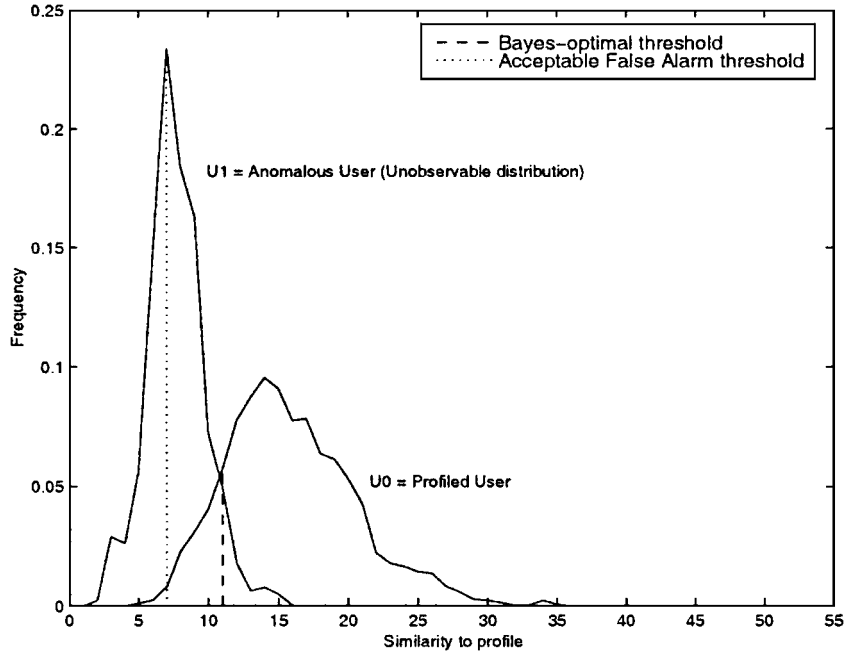

Figure 3. Comparison of unweighted Bayes-optimal decision boundary and acceptable false alarm rate boundary. The rightmost curve (user U0) represents the profiled user.

the false alarm rate. This yields a Neyman-Pearson hypothesis test (Casella & Berger, 1990), for a non-parametric distribution of the form:

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

where v is the similarity of a sequence to be classified to a particular profile, D, 1 denotes "normal", 0 denotes "anomalous", and r is the specified false alarm rate. The test, as given above, does not uniquely determine decision thresholds. We implement this test by selecting two decision thresholds, $t_{\max}$ and $t_{\min}$, based on the upper and lower $r/2$ quantiles of the similarity distribution observed on a "parameter-selection" data set that is independent of both training and testing data. Similarity values falling between the decision thresholds are labeled normal and those falling outside are labeled abnormal. The lower threshold, $t_{\min}$, detects sequences that are too different from known behaviors, while the upper threshold, $t_{\max}$, detects sequences that are improbably *similar* to historical behaviors, perhaps indicating a replay attack.[1]

The parameter r selects the width of the acceptance region on the similarity-to-profile axis. Thus, it encodes the tradeoff between false alarm and false accept error rates. A smaller value of r yields a wider acceptance region and corresponds to a lower false alarm rate. Simultaneously, however, anomalous values have a greater chance of falling into the broader acceptance region, so false accept rates *rise*. The choice of a particular point on the error rate tradeoff curve will depend on site-specific factors such as security policy and the estimated cost of each error class.

3.3. The hidden Markov model representation of anomaly detection

In this section, we describe the use of hidden Markov models as anomaly detection sensors. HMMs are general statistical time-series models that encompass as special cases time-independent multinomial models, fully observable Markov chains, dynamic time warping-based models (Juang, 1984), and edit distance-based models. A HMM is a stochastic finite-state model with separate distributions over output (observable) symbols for each ("hidden") state (Rabiner, 1989). A fixed HMM establishes a distribution over strings from its output alphabet, and the likelihood of a single string can be evaluated via the "forward-backward" algorithm—a dynamic programming procedure that successively evaluates state occupation probabilities for each symbol in the string.[2] The parameters of a fixed-structure HMM are learned from data via the Baum-Welch algorithm, an expectation-maximization procedure that interleaves estimating state occupation probabilities with maximum likelihood parameter estimation based on state occupation estimates.

The general flow of information for the HMM based detector is the same as that given for the IBL sensor in figure 1, but the profile is now an HMM constructed to model the user's historical behaviors via the Baum-Welch algorithm and the similarity measure is now sequence likelihood, evaluated by the forward-backward algorithm.[3] Unlike the IBL model, temporal relations are explicitly represented in the HMM by the state transition matrix, though they are limited to a fixed history depth by the Markov property. The smoothing

function ($F()$) is subsumed by the sequence likelihood evaluation process, but the classification model is still a threshold test (albeit on the log-likelihood space, rather than on the similarity space employed by the IBL classifier). We describe the method in detail below.

3.3.1. Notation. We employ a variant of Rabiner's HMM notation (Rabiner, 1989) in which q_t and O_t are variables denoting the HMM state and observed output at time t, respectively, $\bar{q}$ and $\bar{O}$ denote the complete sequence of states and outputs for the whole period of observation, π_i is the prior probability of state i, a_{ij} is the probability of transitioning from state i to state j, and $b_i(o)$ is the probability of generating output symbol v_o when the HMM is in state i. The matrix forms of the probabilities are Π, A, and B, respectively, and the set of all HMM parameters, $\{\Pi, \mathsf{A}, \mathsf{B}\}$ is denoted θ. The number of hidden states in the model is K and the size of the alphabet of observable symbols is $|\Sigma|$. While the particular state that the model is in at time t is denoted q_t, the states themselves are labeled $S_1, S_2 \ldots S_K$. Similarly, the output symbols are labeled $v_1, v_2 \ldots v_{|\Sigma|}$ and the particular symbol observed at time t is O_t.

3.3.2. HMMs as sequence data classifiers. The task of employing hidden Markov models as temporal classification systems can be framed in at least three different ways. One popular method for multiclass problems is to identify the class labels with the hidden states of a single model. The state sequence inferred from observed data via the Viterbi algorithm (Rabiner, 1989) then constitutes the classification of the temporal sequence data. Such an approach has been employed in, for example, speech recognition (Rabiner & Juang, 1993), positional tracking and prediction in user modeling (Orwant, 1995), and fault monitoring (Smyth, 1994a, 1994b). Smyth describes this approach as *discriminative*, viewing the classification problem as one of estimating the probability of class labels given the data and model parameters, $p(\bar{q} \mid \bar{O}, \theta)$ (Smyth, 1994b). This approach makes the assumptions that the class labels (states) are mutually exclusive and exhaustive. While the first condition certainly holds for the anomaly detection domain—any given input token can be generated by only a single user—the latter poses a considerable difficulty. In the anomaly detection domain we clearly have examples of the valid user's behavioral characteristics, but we lack examples of the behaviors of hostile users or intruders. Even given examples of hostile behaviors, however, the problem of demonstrating that our training set is exhaustive may be difficult at best.

In the fault detection domain, Smyth addresses the question of unobserved classes by adding an extra "catch-all" state to the model and augmenting the discriminative model with a *generative* model. A generative model views the HMM as a data generator and estimates observation likelihoods, $p(\bar{O} \mid \theta)$, via the forward step of the forward-backward algorithm (Rabiner, 1989). Class probabilities can be derived from instantaneous observation probabilities, $p(O_t \mid q_i)$, via Bayes' rule. The hybrid of discriminative and generative approaches allows estimation of class probabilities for an auxiliary state modeling unobserved data. The combination of the two classes of models involves prior distribution assumptions about the likelihood of the data under the unknown class.

In this work, we take a different approach to the classification problem. We will first describe a general classification framework for an $N + 1$ class domain (one of which is an "other" or "unknown" class), and then will restrict it to our binary classification domain.

Similar to the generative approach, we employ estimations of data probabilities via the forward-backward algorithm, but rather than associating class labels with model states, we associate class labels with individual *models*. Model probabilities can be evaluated from posterior observation probabilities via Bayes' rule:

$$p(\theta \mid \bar{O}) = \frac{p(\bar{O} \mid \theta)p(\theta)}{p(\bar{O})},$$

where $p(\bar{O})$ is a normalizing factor that is identical for all classes. The model prior probability, $p(\theta)$, can be selected by domain knowledge, but we take it here to be a non-informative prior (i.e., a uniform probability distribution for a finite set of models). For an N class problem, an observational sequence is assigned the class label corresponding to the maximum likelihood model,

$$\text{class}(\bar{O}) = \underset{i \in 1 \ldots N}{\text{argmax}}\{p(\theta_i \mid \bar{O})\}.$$

Effectively, we are assessing the likelihood that each model generated the sequence in question and selecting the model with the highest likelihood. This framework allows us to assign only a single label to an entire observational sequence, but gives us the freedom to assign "unknown" class labels. Any sequence judged insufficiently likely with respect to all known models can be labeled "unknown". Similar "model-class" approaches have been widely applied in the speech recognition community (Rabiner, 1989). Orwant used a related framework to determine a user's current behavioral state (e.g. "idle", "writing", or "hacking") but employed manually constructed models for each class and interconnected the class models into a "meta-HMM" from which classes were predicted via the Viterbi algorithm (Orwant, 1995). In our binary domain, we construct a single model of the user and data not fitting that model well are assigned the "anomaly" label. The model prior probability expresses our knowledge about the incidence rate of impostors, and we take it to be 0.5 in this work (i.e., data from an impostor is as likely to occur as is data from the valid user). This is (hopefully) an unrealistic assumption, but we have not been able to locate solid figures on real incident rates with which to select a prior.

The choice of K is important, as it affects the potential descriptiveness of the HMM. In the discriminative and generative forms of HMM classification, the domain provides us with an appropriate value of K (the number of classes present in the data, and possibly one or more "unknown" states). In the model-class framework, however, the classes are not directly associated with model states so we must seek either domain-specific knowledge to help choose K (e.g., some estimate of the natural number of distinct behavioral classes present in the data) or employ an empirical search. We examine the latter method in the experimental section of this article.

3.3.3. Sequence labeling. Under the model-class framework outlined above, we construct a *single* HMM, θ_v, to model the observed behavioral patterns of the valid user. The likelihoods of incoming data sequences are evaluated with respect to θ_v and those judged insufficiently likely via a threshold test are labeled as anomalous. The value of this "minimum acceptable likelihood" is denoted $t_{\min}$. As in the IBL sensor, to thwart replay attacks

we introduce an upper threshold, $t_{\max}$, which is used to flag data that are too similar to historical behaviors. The thresholds, $t_{\min}$ and $t_{\max}$ are chosen from the upper and lower $r/2$ quantiles of the non-parametric distribution of observation likelihoods on an independent, parameter-selection, subset of the training data. Again, the parameter r corresponds to the acceptable false-alarm rate.

3.3.4. Sequence alignment. As noted above, the model-class framework assigns class labels only to entire sequences, yet we wish to be able to label arbitrary subsequences of the observed data stream. We can, of course, run the forward-backward likelihood estimation algorithm between every possible pair of subsequence start, s, and termination, t, time steps. This turns out to be computationally expensive, as the complexity of the F-B algorithm is $O(K^2W)$ for a time sequence of length W.[4] Merely to consider all fixed-length subsequences ($t - s = W$ for some fixed W for all t) within a total data sequence of length T requires $O(K^2W(T - W))$ time. This becomes prohibitive for the subsequence lengths of interest in this domain ($W > 50$).

Instead we employ the approximation algorithm obtained by considering the endpoint state transitions (those at time steps s and t) to be statistically uncoupled from their adjacent states (those at time steps $s - 1$ and $t - 1$). That is,

$$p(O_s O_{s+1} \ldots O_t) \approx \frac{p(O_1 \ldots O_T)}{p(O_1 \ldots O_{s-1})p(O_{t+1} \ldots O_T)}$$

for $1 < s < t < T$. Because of the exponential decay of state influence in the Markov formulation, this approximation is reasonably good for large W. For example, a comparison of the approximated sequence log-likelihood to the exact value for one of our tested users at $W = 100$ (the value used in our empirical investigations, Sections 4.5 and 5.2 revealed that the approximated value had a mean deviation of only 0.8% and a median deviation of only 0.46% from the true value (indicating that the deviations are skewed towards 0). Thus, this approximation allows us to consider all fixed-length subsequences from a global temporal sequence of length T in time $O(K^2T + (T - W))$, with only a marginal loss in precision.

4. Empirical analysis: Base-line systems

In this section we describe the performance criteria we use to evaluate models and the experimental design under which measurements are performed. We examine the performance of one class of IBL and HMM user models, which we shall denote the *base-line models*. The base-line models will be used as standards for comparison when we investigate model scaling issues in Section 5.

For the IBL sensor, the interpretation of "base-line model" is straightforward: the model containing all available training data. For the HMM models "base-line" is less well defined. HMMs subsume, as boundary-cases, both fully-observable Markov chain models and "atemporal" models that characterize sequence data as independent observations drawn from a multinomial distribution (i.e., $K = 1$), as well as a wide spectrum of "standard" HMMs in between. Among these, it is unclear which can be considered most nearly comparable to the IBL base-line model. We have, therefore, examined a range of such models (described in

detail in Section 5.2). For the sake of comparison, we wish to select an HMM base-line that bears some similarity to the IBL base-line model. Thus, we select as our HMM base-line that model with a parameter cardinality closest (among those tested) to our base-line IBL models. Under the experimental design that we employ here, the base-line IBL models contain 990 vectors of 10 symbols each and the base-line HMMs contain 30 hidden states and an average alphabet size of 250 yielding a total space of 8430 parameters ($K^2 + K|\Sigma| + K$).

4.1. Performance criteria

We apply three criteria to evaluate the performance of anomaly detection systems. In addition to the traditional *accuracy* measurements, we argue that the *mean time to generation of an alarm* is a useful quantity to consider and that when considered as a focus of attention component, the degree of *data reduction* achieved by the sensor is important.

The ultimate goal in the anomaly detection task is to identify potentially malicious occurrences while falsely flagging innocuous actions as rarely as possible. We shall denote the rate of incorrectly flagging normal behaviors as the *false alarm* rate and the rate of failing to identify abnormal or malicious behaviors as the *false acceptance* rate. Under the null hypothesis that all behavior is normal, these correspond to Type I and Type II errors, respectively. The converse accuracy rates are referred to as the true accept (ability to correctly accept the profiled user as normal) rate and the true detect (ability to correctly detect an anomalous user) rate. Although it is necessary that the false alarm rate of an *entire* anomaly detection system be low, that is not the primary goal of our sensor, which functions as a focus of attention mechanism. A FOA sensor may be able to accept high numbers of false alarms, which can be removed by higher-level decision processes in the classification hierarchy, but any "notable events" (i.e., anomalies) that it misses will never be available to higher-level components.

As pointed out by Fawcett and Provost (1999), detection accuracy alone does not reveal the complete story in domains of this type. A second issue of importance is *time to alarm* (TTA). We wish the time to alarm to be short for anomalies so that potentially hazardous events can be dealt with quickly and before doing much harm, but long for the valid user so that normal work is interrupted by false alarms as seldom as possible. We define the TTA to be the mean run-length of "normal" classifications (i.e., the mean time between alarms), as a measure of the maximum time an impostor can escape detection. We report TTA values in token counts because this value is more nearly correlated with work accomplished (or damage inflicted) than is wall clock time. TTA is an average figure and can achieve fractional values.

4.2. Data source and collection

In this work we examine user behaviors as displayed in UNIX command shell data. We demonstrate the behavior of the detection system on the task of differentiating different authorized users of UNIX hosts.[5] In this framework, an anomalous situation is simulated by testing one legitimate user's command data against another legitimate user's profile. This framework simulates a subset of the possible misuse scenarios—that of a naïve intruder gaining access to an unauthorized account. Though this scenario doesn't address skillful

attackers who are fully aware of the capabilities of the anomaly detector and are deliberately attempting to defeat it, it *does* address an important class of real attack scenarios. In recent years, a proliferation of automated attack and intrusion scripts (so-called "kiddie scripts") have allowed a growing number of unsophisticated would-be system crackers to penetrate systems far beyond their own native capabilities. Real incidence numbers are difficult to come by, but security experts informally estimate that a majority of system penetrations on the Internet today may be perpetrated by such naïve attackers. Thus, progress on even the naïve attacker scenario is valuable to the security community.

We examine system performance on UNIX shell data collected from two different user populations. The first group consists of eight different UNIX users at Purdue University, monitored over the course of more than two years. The amount of data available varied among the users from just over 15,000 tokens to well over 100,000 tokens, depending on their work rates and when each user entered and left the study. Because of computational constraints and for testing uniformity, we employed a subset of 10,000 tokens from each user, representing approximately four months of computer usage.

The second user population is a subset of the 168 users monitored by Greenberg at the University of Calgary (Greenberg, 1988). Our experimental method requires 2,000 tokens per test fold per user and 98 of the Calgary users are represented by this amount of data. The Calgary users were drawn from four populations: 33 computer scientists, 25 experienced programmers, nine non-programmers, and 31 novice programmers. The Calgary data are independent of the Purdue data and were not examined during the development of the system described here. Because of differences in shell and data collection technique, the Purdue and Calgary user populations are not directly comparable to each other, but members of each population can be compared to other members within that population.

4.3. Experiment structure

Because user behaviors change over time, the effective lifetime of a *static* user profile, as is employed in the work described here, is limited. Thus, we have constructed experiments to evaluate the detector's performance over a limited range of future activities. Within the Purdue data, each user's 10,000 tokens were divided into five independent subsets (or folds) which were, in turn, split into independent train (1,000 tokens), parameter selection (500 tokens), and test (500 tokens) data sets. For each fold, the user profile was constructed from the training data and the classification thresholds ($t_{\max}$ and $t_{\min}$; see Section 3.2) were selected with respect to the parameter selection data. The Calgary data were treated similarly except that only a single fold was constructed from the 2,000 tokens available from each user. Because the Baum-Welch training algorithm for HMMs is subject to becoming trapped in poor local maxima of the parameter likelihood space, we employed multiple restarts for the HMM experiments. Each HMM experiment was repeated from five randomly drawn starting locations, and results were averaged across the five restarts. The same data splits were used for the IBL and HMM experiments.

From each training set, a profile was constructed with $l = 10$ (the fixed sequence length for the similarity measure) and $W = 100$ (the window length for the noise-suppression filter). These values were chosen based on prior empirical work in this domain (Lane, 1998; Lane

& Brodley, 1997b). The resulting profile was tested against the corresponding test set for each user (a total of 8^2 test pairings per fold for the Purdue data and 98^2 test pairings for the Calgary data). A "SELF" test pairing—testing the profiled user's data against his or her own profile—allows us to examine false alarm rates while a "non-self" or "OTHER" pairing allows us to examine false accept rates.

The acceptable false alarm rate, r, determines how the classification thresholds, $t_{\max}$ and $t_{\min}$, are set and has a substantial impact on the tradeoff between false alarm and false accept errors. Because the notion of "acceptable" false alarm rate is a site-dependent parameter, we wish to characterize the performance of the system across a spectrum of rates. We took $r \in \{0.5, 1, 2, 5, 10\}\%$, which yields a performance curve for each profile/test set pair. This curve, which expresses the tradeoff between false alarm and false accept errors with respect to r, is known as a Receiver Operating Characteristic (ROC) curve (Provost & Fawcett, 1998). An ROC curve allows the user to evaluate the performance of a system under different operating conditions or to select the optimal operating point for a given cost tradeoff in classification errors.

All statistical results reported in this article were calculated with a Wilcoxon signed-ranks test at a significance level of $\alpha = 0.01$ (Sheskin, 1997).

4.4. IBL base-line results

We will begin by examining the performance of the IBL sensor on a single individual's profile. This will allow us to examine some of the features of the domain and data in detail. We will then present full results for all profiles.

4.4.1. Single profile results. Example results for a test fold of a single profile (that of USER3) are shown in figure 4. Accuracies for USER3's profile are given in (a) and

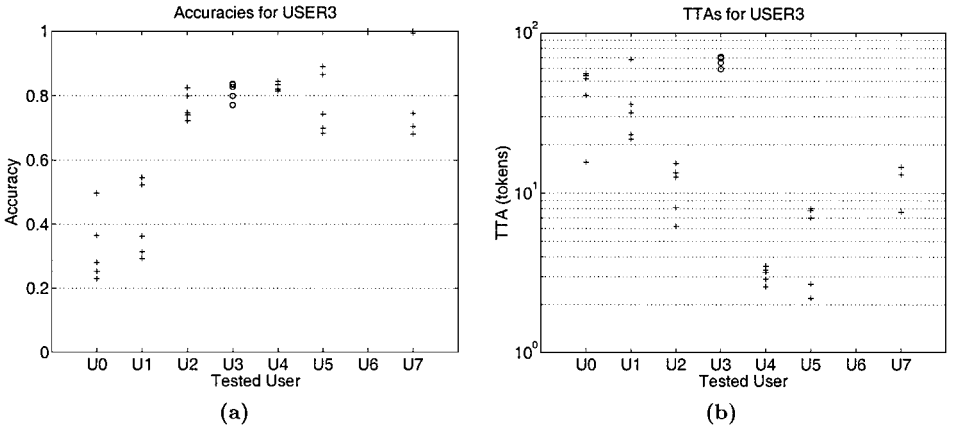

Figure 4. (a) Accuracy and (b) time-to-alarm results for the IBL base-line system, single fold, for USER3's profile. Each column shows results for one test set against the profiled user. "+" symbols denote adversaries, while "o" symbols denote the profiled user. Opponent U6, with accuracy 1, has mean TTA 0, which is not representable on a logarithmic axis.

time-to-alarm (TTA) values are shown in (b). While the accuracies are linearly scaled on the range [0 . . . 1], the TTAs are logarithmically scaled. Each column in these plots displays the accuracy results for a single test set when tested against the profile. When the test set originates with the profiled user (i.e., USER3 tested against Profile 3), the results indicate the ability to correctly identify the valid user (true accept rate, in (a), or mean time between *false* alarms in (b)). This condition is denoted with an "o" symbol in the plot. When the test set originates with a different user (e.g., USER5 tested against Profile 3), the results indicate the ability to correctly flag an anomalous condition (true detect rate, in (a), or mean time between *true* alarms in (b)). This condition is denoted with a "+" symbol on the plot. In all cases, accuracy and mean time to alarm is increasing in the positive direction on the Y axis (while high accuracy is uniformly desirable, a *long* TTA is desired for the profiled user but a *short* TTA is desired for the impostor). The spectrum of results in each column is generated by testing at different values of r, the acceptable false accept rate, as described in Sections 3.2.5 and 4.3. Because r encodes the size of the acceptance region, it yields a tradeoff in detect versus accept accuracies. The smallest value of r tested ($r = 0.5\%$) yields the widest acceptance region and corresponds to the highest (most accurate) point on the true accept column (USER3) in (a). But because the acceptance region is wide, more anomalous points fall into it and are accepted falsely. Thus, this value of r corresponds to the *lowest* accuracy in each of the true detect columns (USER$\{0$–$2, 4$–$7\}$).

Profile 3 was selected to highlight a number of points. First is that accuracy and TTA are highly sensitive to the particular opponent. USER1 and USER4, for example, display quite different detection accuracies and their TTAs differ by almost an order of magnitude. Second, although the acceptable false alarm rate parameter, r, was tested across the range 0.5%–10%, all of the observed false alarm rates are higher than these settings (16.3%–23.0%). This is a result of the training and parameterization data failing to fully reflect the behavioral distribution present in the testing data. Because the user has changed behaviors or tasks over the interval between the generation of training and testing data, the profile does not include all of the behaviors present in the test data. This phenomenon is exacerbated by the batch-mode experimental setup used here. In tests of the online version of this system we have found that continuously adapting to the user's behaviors (thus shortening the delay between training and testing) improves true accept accuracy (Lane & Brodley, 1998).

An example of another source of false accept error is demonstrated in figure 5 (drawn from a different part of the experiment). Here, the profiled user (USER6) and USER5 have many behaviors in common—mostly "generic" account maintenance such as directory creation and file copy and remove operations. This high degree of similarity is reflected in the substantial overlaps in the similarity distributions, making differentiation impossible within this space. By contrast, USER3 was engaged mainly in programming and writing during this time. There are two possible sources for the degree of overlap between USER5 and USER6. First, the underlying observations do not encode sufficient information to distinguish the two users. Many other data sources are available for user profiling and could be used in conjunction with the techniques presented here in an operational security system. The second, and more fundamental, source of error is in the similarity measure itself. The measure presented in this article is fairly coarse (having only $O(l^2)$ possible values for

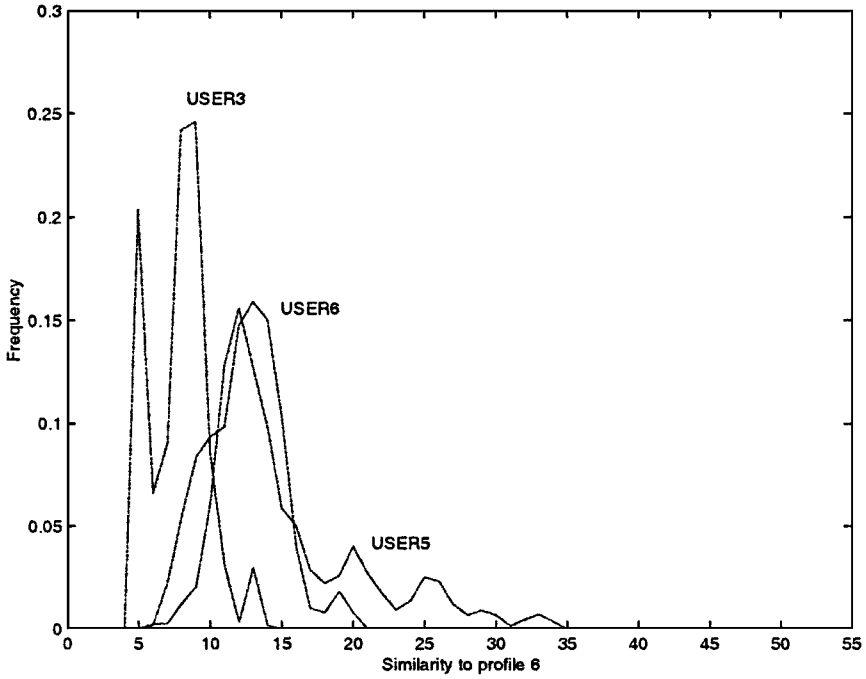

Figure 5. False accept errors: USER5's data bears high resemblance to the profiled user's (USER6).

a sequence length of l), and models only a single type of temporal interaction. We are currently investigating more sophisticated similarity measures.

Figure 4 also demonstrates that accuracy and TTA are measuring slightly different quantities. In particular, observe that while USER2 and USER4 demonstrate quite different TTAs, their accuracies fall in approximately the same range. Clearly, if the alarms were uniformly distributed within the classification stream, the two performance measures would be equivalent. We have observed, however, that false alarms tend to occur in runs while true alarms are more sporadically distributed. This is desirable because when false alarms occur in a run they can be examined and dismissed as a group, while when true alarms occur only a single alarm may be needed to catch the attacker.

4.4.2. Results for all profiles. A visual representation of performance results for the baseline system tested on all Calgary profiles at a false alarm rate of $r = 0.01\%$ is given in figure 6. The histogram displays for the Purdue users are similar, though coarser. These histograms display the overall accuracy ((a) and (c)) and mean time-to-alarm ((b) and (d)) performance of the sensor when profiling the Calgary users. Statistically, the median true acceptance accuracy on the Purdue data is 78.3% and the median true detection accuracy is 85.2%. The corresponding median TTAs are 71.0 tokens (valid user) and 2.9 tokens (impostors).

The net result is that the sensor is performing well both at detecting impostors and recognizing the profiled user, though performance is clearly better on the former task.

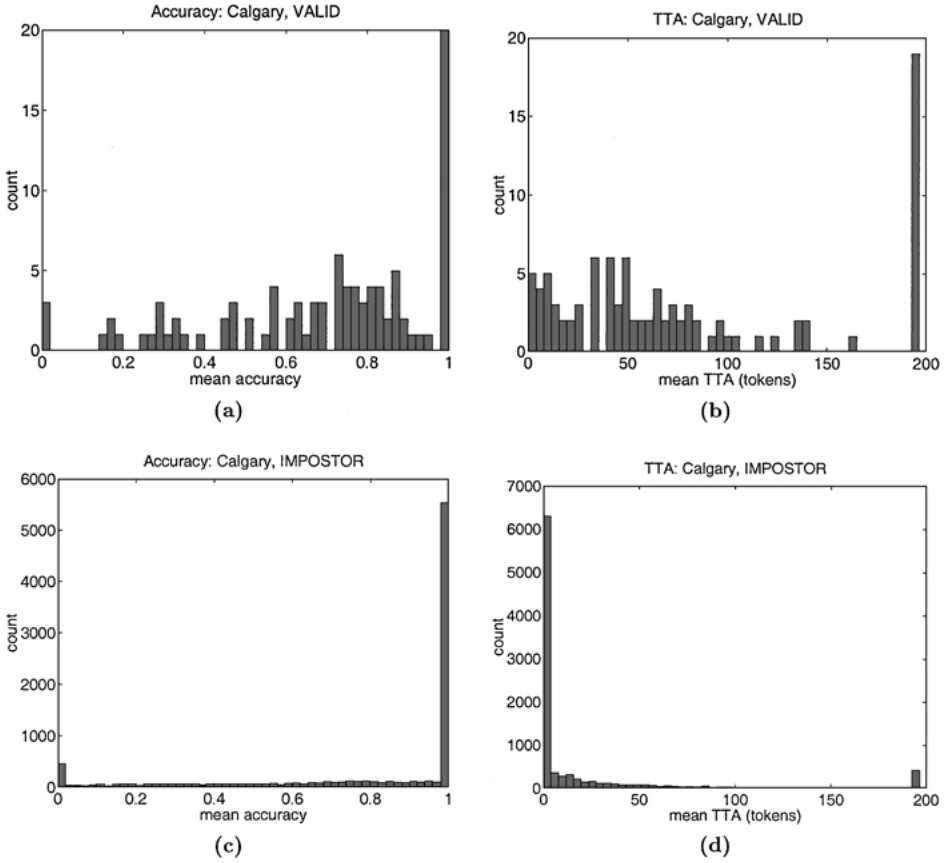

Figure 6. Histograms of the base-line system ((a) and (c)) accuracies and ((b) and (d)) TTAs on the Calgary data. The horizontal axes give the value achieved on that measure and the vertical give the number of test sets displaying that level of the measure. Values for the profiled user are shown in (a) and (b) while those for impostors are given in (c) and (d).

While this balance might be undesirable in a standalone detection system, it is promising in the context of a focus of attention sensor (see Section 2). In a FOA context, the sensor has achieved approximately a 70% level of data reduction on non-anomalous data in exchange for detecting almost 80% of the anomalous events. If we relax the acceptable false alarm rate to $r = 10\%$, we improve mean impostor detection to 83% at a cost of dropping to about 63% data reduction on non-anomalous events.

4.5. Hidden Markov model base-line results

A comparison of aggregate HMM results for a sensor with $k = 30$ hidden states to base-line IBL sensor results is given in Table 1. It appears that the two classes of sensors are very close in ability to recognize the profiled user (SELF) and, in fact, they are statistically

Table 1. HMM sensor accuracies and mean time-to-alarm values compared to the corresponding IBL sensor values at an acceptable false alarm rate of $r = 0.5\%$. Statistically superior results are highlighted.

	IBL		HMM	
	SELF	OTHER	SELF	OTHER
		Calgary		
ACC	69.2%	80.0%	69.5%	**84.3%**
TTA	79.8	20.5	82.3	**16.5**
		Purdue		
ACC	72.4%	72.7%	71.1%	73.4%
TTA	85.0	24.8	83.8	25.7

indistinguishable both in true accept accuracy and in mean time-to-false-alarm. In ability to distinguish impostors, however, the HMM sensor is statistically significantly better in both accuracy and TTA for the Calgary data. These results are balanced by two factors. The first is run time: classification time for the IBL model is linear in size of the model and training time is constant while classification time for the HMM sensor is quadratic in model size and training time is even greater.[6] Runtime may be a precious resource for a security system—especially if the anomaly detection sensor is only one component at the bottom tier of a hierarchical classifier.

The second mitigating factor is displayed in figure 7. These scatter plots show the relative performance of the two systems—each mark corresponds to the HMM (x-axis) and IBL (y-axis) accuracies for a particular Calgary test set. The diagonal line is the iso-performance line and points to the right of it indicate superior performance by the HMM sensor while points above it indicate superior IBL performance. The detection superiority of the HMM sensor is visible as a higher density of points to the right of the iso-performance line than above it

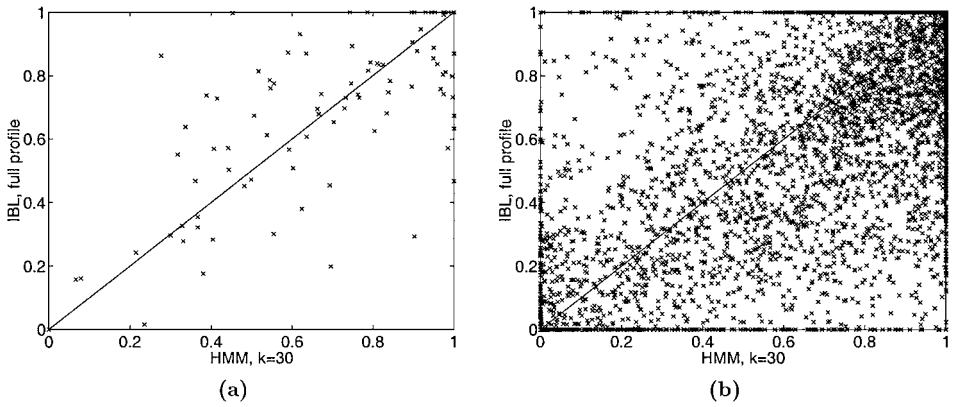

Figure 7. Scatter plots of IBL versus HMM (a) true accept/SELF and (b) true detect/OTHER accuracies (corresponding to the aggregate figures for the Calgary data given in Table 1).

in (b). The tendency is slight, however, and there are a great many points well distributed on the other side of the iso-performance line. Indeed, there are a number of cases in which one system falls to zero accuracy while the other attains large non-zero values (along the left and bottom edges). Similarly, though aggregate true accept performance is not statistically distinguishable for the two systems, the variance is high on a per-user basis. Thus, neither system dominates the other in all cases. The ideal classification strategy, therefore, is to either locate an a priori method for choosing the model class appropriate to a given user or to combine the two methods in a single classifier system (e.g., via voting or as different focus-of-attention sensors in a multi-tier classifier).

5. Model scaling

In this section we examine the impacts of manipulating the number of trainable parameters present in the user models. For the IBL models, this amounts to reducing the number of instance vectors stored in the user profile, while for the HMMs it entails changing the number of hidden states in the model. We examine two classes of methods for selecting appropriate instances to preserve in the IBL profiles and we investigate a range of values of K for the HMM user models. We give empirical analyses for all models and tie the performance of the scaled models back to those of the base-line models examined in Section 4.

5.1. IBL model scaling: Clustering

A widely acknowledged weakness of instance-based learning algorithms is the large data storage requirement for accurate classification. In an operational setting, data reduction is critical as the size of the profile directly impacts the time required for classification. A number of techniques have been examined for reducing this memory overhead, but most of these require independent binary- or multi-class data to evaluate the utility of each instance (Dasarathy, 1991; Wilson & Martinez, 2000). We have previously examined instance selection methods of data reduction for this domain, in which the utilities of instances within the profile are evaluated with respect to independent data drawn only from the profiled user and low utility instances are discarded (Lane & Brodley, 1999). We found that the instance selection techniques examined there all suffered from a form of misgeneralization as more data were removed from the profile. Specifically, the models formed by instance selection generated more alarms (both false and true) than did the base-line models. Effectively, the instance selection techniques were learning the concept that "nobody is the valid user".

An alternate method of reducing data storage is to modify the representation of sets of points within the data space. For example, Salzberg (1991) represented sets of points as hyper-rectangles, while Domingos (1995) induced rules that cover subsets of the instance base. In this section, we examine the use of clustering methods to locate correlations within the user profile (i.e., without respect to any independent data) and to remove redundant points from the profile.

A cluster can be represented by a single exemplar instance, or center point, which is the instance having the smallest distance to all other instances in the cluster. By discarding all

other elements of the cluster, substantial space and time savings can be realized. Although the practical effect of this process is the same as that of the instance selection methods described above, the clustering process employs knowledge about the relationships among elements within the profile while the pruning methods employ only knowledge about the relationship between individual profile elements and the "external" parameter selection data.

One popular class of clustering algorithms is based on the Expectation-Maximization (EM) procedure (Moon, 1996). These methods attempt to simultaneously maximize an optimality criterion across all clusters through gradient descent on the cluster likelihood space. The basic process is to assign all points to clusters and evaluate the optimality criterion under that labeling. The evaluation yields a parameter set that is used to reassign points to clusters. This basic loop is repeated until the optimality criterion converges to a stable point. A common implementation of this process is the K-means algorithm, or its discrete analog, K-centers. In these algorithms, clusters are parameterized by their centroid (continuous) or center (discrete) points and radii, and the search locates K clusters.

Clustering methods based on EM are popular because they are general and often highly effective. However, when many local optima are present in the likelihood space, the quality of the solution produced can be very sensitive to the initial assignment of points to clusters. A larger difficulty for the anomaly detection domain is that K, the number of clusters to be sought, must be known a priori yet it is not clear how to determine the number of "natural" clusters in the user behavioral data. Furthermore, the convergence rate of these methods is not guaranteed and is often related to the number of clusters sought. For a large K, the search time for a stable solution can be prohibitive.

As a response to these difficulties, we propose a greedy clustering algorithm that builds individual clusters consecutively, attempting to minimize the criterion:

$$\text{val}(\mathbf{C}) = \frac{\sum_{x \in \mathbf{C}} \sum_{y \in \mathbf{C}} \text{Dist}(x, y)}{|\mathbf{C}|^2}$$

for each cluster $\mathbf{C}$. This measure is a generalization of the mean inter-cluster distance often employed for clustering (Fukunaga, 1990) in continuous domains. From an initial seed point, the cluster is grown incrementally by including the point that increases val($\mathbf{C}$) the least, until the halting criterion is satisfied. Growth is halted when the value of the cluster's criterion function reaches a local maximum. Because, in some cases, the cluster value monotonically approaches $\text{Sim}_{\max}$, the halting criterion we actually use is that the first derivative of val($\mathbf{C}$) be within ϵ of 0 for some (empirically selected) value of ϵ. As each sequence is added to a cluster, it is removed from the set of available sequences. When the cluster is complete, we define the center of the cluster, $\mathbf{C}_{\text{cent}}$, to be the point possessing the minimum total distance to all other points in $\mathbf{C}$. The similarity between a sequence, X, and a cluster is then $\text{Sim}(X, \mathbf{C}_{\text{cent}})$.

In practice, we have found that this cluster selection algorithm is too lenient—it accepts points that decrease the cluster's effectiveness in classification. We solve this in a manner analogous to the pruning process employed in decision tree learning (Quinlan, 1993). After growing a single cluster to completion according to the halting criterion, the clustering algorithm removes outlying points and returns them to the pool of available sequences (so that they have the possibility of contributing to different clusters). Our pruning function

removes points from the cluster that fall outside the cluster mean radius—i.e., points whose distance to the center is greater than the mean distance to the center of all points in the cluster. Points falling within the mean radius are discarded and the final cluster is represented only by its center and mean radius. We realize substantial space savings by replacing the set of all cluster elements with the single center point.

The complete clustering algorithm is structurally similar to the single cluster construction algorithm. We sequentially select individual clusters by their ability to maximize the analog of mean intra-cluster distance:

$$\text{val}\{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n\} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} \text{Dist}(\mathbf{C}_{i,\text{cent}}, \mathbf{C}_{j,\text{cent}})}{n^2}.$$

In this case, we found the single cluster halting criterion to be ineffective because, typically, all of a data set's points were exhausted before the derivative of the intra-cluster distance approached 0. When we allowed the clustering process to absorb all available points, many of the clusters were found to either not contribute to classification accuracy or to be actively harmful.[7] Instead, we halt the clustering process when the minimum inter-cluster value of all current clusters falls below a threshold, C. C determines when the clustering process will be halted and how many clusters will be constructed. A small value of C produces many clusters, while a large one allows many of the original profile points to remain. We examine the effects on performance of the choice of the parameter C below.

5.1.1. Comparison of clustering techniques. We examined the performance of the K-centers and greedy clustering methods for storage reduction under the experimental conditions described in Section 4.3. The additional parameters examined were K, the number of clusters formed by the K-centers method, and C, the global halting criterion for the greedy clustering method. For the greedy clustering method, we examined $C \in \{0.25, 0.5, 0.75\}$ and used a fixed $\epsilon = 0.005$.[8] For K-centers, we ran the search to 10,000 cycles or convergence, and examined $K \in \{50, 75, 100, 125, 150\}$ instances. Additionally, for the Purdue data we examined $K \in \{324, 640\}$ and for the Calgary data we examined $K \in \{429, 652\}$. These last were chosen to correspond to the average data retention rate (number of clusters plus outliers) achieved by the greedy clustering algorithm at $C = 0.25$ and $C = 0.5$ for the two user populations. Typically, the K-centers procedure expired its 10,000 search cycles before converging at these values of K. We did not attempt to match a value of K to the greedy system at $C = 0.75$ because the latter achieves a very low degree of data compression—retaining on average 979 of the original 991 instances.

Table 2 displays the model compression ratios achieved by the two clustering methods. This value is the ratio of the number of instances removed from the profile by the clustering process to the original number of instances. The ratios given for the greedy method include both the retained cluster centers and retained outlier points. The compression level produced by the K-centers method is constant for a given K. It is clear from this table that the K-centers method can achieve an arbitrarily high compression level simply by selecting a smaller value of K (up to $K = 1$). The greedy method, however, is bounded in the degree of model compression it can achieve. The value of C controls the amount of search and, because of the greedy nature of cluster formation, the number of clusters located is strictly

Table 2. Model compression ratios for the evaluated parameter settings of the greedy and K-centers clustering algorithms. The compression rates for $K = 25$–150 are the same for both the Purdue and the Calgary data.

	Purdue	Calgary
	Greedy	
$C = 0.25$	67.3%	56.7%
$C = 0.50$	35.4%	34.4%
$C = 0.75$	1.1%	1.0%
	K-centers	
$K = 25$	97.5%	
$K = 50$	95.0%	
$K = 75$	92.4%	
$K = 100$	89.9%	
$K = 125$	87.4%	
$K = 150$	84.9%	
$K = 324$	67.3%	–
$K = 429$	–	56.7%
$K = 640$	35.4%	–
$K = 652$	–	34.4%

non-decreasing as C is lowered until the pool of outliers is completely exhausted. Thus, the compression level of the greedy clustering method is limited by the maximum number of clusters that it forms as C is decreased. In this data, greedy is limited to roughly 75% compression. Thus, if a very high degree of model compression is desired, K-centers is preferred. Over the compression levels that the two methods have in common, however, other criteria must be used to choose between them.

An important alternative criterion is accuracy, as displayed in Table 3. Here we give the mean accuracies for the base-line system (no model compression) and the greedy and K-centers clustering methods for both user populations. The "SELF" column reports true accept accuracies, the "OTHER" column reports true detect accuracies, and the "TOTAL" column reports total mean accuracy (both true detect and true accept). The values appearing in the table are averaged across r (the acceptable false alarm rate; Section 3.2) as well as across fold and user. The statistical results reported below are invariant across values of r.

The accuracies of the greedy clustering method are very close to those of the base-line system. In fact, for the Purdue data all the results are statistically indistinguishable and for the Calgary data the SELF results are statistically indistinguishable from the base-line. While the OTHER and TOTAL columns for the Calgary data are very close to those for the base-line, the accuracy distributions actually differ at higher-order moments and the base-line system is statistically stronger. We conclude from these results that using the greedy clustering method costs no significant true accept accuracy and in some cases costs no true detect accuracy.

The second important point in Table 3 is that the true detect accuracies ("OTHER" column) for the K-centers algorithm are lower than those for base-line and greedy. It is

Table 3. Mean accuracy values for the base-line system and the tested parameter values of the two clustering methods on both user populations. Values are accuracy percentages.

	Purdue			Calgary		
	SELF	OTHER	TOTAL	SELF	OTHER	TOTAL
			Base			
	69.7	74.6	74.0	66.9	81.1	80.9
			Greedy			
$C = 0.25$	69.8	74.7	74.1	67.3	81.0	80.8
$C = 0.50$	70.1	74.3	73.8	67.0	81.4	81.2
$C = 0.75$	69.6	74.6	74.0	66.9	81.1	80.9
			K-centers			
$K = 25$	70.8	60.6	61.9	66.4	65.0	65.0
$K = 50$	66.9	65.0	65.2	66.5	70.3	70.3
$K = 75$	69.3	67.7	67.9	65.5	73.6	73.5
$K = 100$	68.4	68.4	68.4	66.5	75.4	75.3
$K = 125$	66.2	70.4	69.9	66.4	75.5	75.4
$K = 150$	68.0	69.5	69.3	66.9	76.7	76.6
$K = 324$	70.3	72.8	72.5	–	–	–
$K = 429$	–	–	–	66.7	78.1	78.0
$K = 640$	69.5	72.1	71.8	–	–	–
$K = 652$	–	–	–	67.5	78.1	78.0

unsurprising that accuracy should suffer at the most extreme compression levels. What is more important is that at the more modest levels, K-centers does not achieve the same true detect accuracy as does greedy. At directly comparable levels of compression ($K = 324$ vs. $C = 0.25$ and $K = 640$ vs. $C = 0.5$ for the Purdue data and $K = 429$ vs. $C = 0.25$ and $K = 562$ vs. $C = 0.5$ for the Calgary data), K-centers has statistically significantly worse performance on true detect accuracies than does greedy. On true accept accuracy, however, K-centers is statistically indistinguishable from base-line or greedy across the board.

The question of how true accept rates can remain unchanged while true detect rates degrade bears some examination. Recall from Section 3.2.5 that the decision thresholds, $t_{\max}$ and $t_{\min}$, are selected from the self-similarity distribution of the valid user's behaviors. This distribution is estimated from similarity measurements between out of sample "parameter selection" data (Section 4.3) and the *clustered* profile. If the profile is no longer representative of the scope of the user's behaviors, the self-similarity distribution will be skewed and the decision thresholds will be chosen poorly. The resulting thresholds may still identify the valid user well (if the parameter selection data reflects the test data well) but will also be more likely to accept impostor data.

Table 4 displays the mean time-to-alarm values for the same evaluation points as are given in Table 3. The "SELF" column reports mean time to generation of a false alarm while the "OTHER" column reports mean time to generation of a true alarm. There is no "TOTAL" column in this table because SELF and OTHER are interpreted differently and

Table 4. Mean TTA values for the base-line system and the tested parameter values of the two clustering methods on both user populations. Values are mean times in token counts.

	Purdue		Calgary	
	SELF	OTHER	SELF	OTHER
		Base		
	76.8	22.5	74.7	18.9
		Greedy		
$C = 0.25$	79.3	21.5	74.4	18.8
$C = 0.50$	79.7	22.6	74.0	18.5
$C = 0.75$	75.6	22.4	74.8	18.9
		K-centers		
$K = 25$	82.4	39.7	77.9	44.7
$K = 50$	74.7	33.1	78.5	35.2
$K = 75$	79.3	30.7	72.6	30.0
$K = 100$	73.1	30.1	76.0	26.9
$K = 125$	73.0	24.9	72.9	26.7
$K = 150$	73.4	25.5	76.6	25.0
$K = 324$	77.3	23.4	–	–
$K = 429$	–	–	75.5	22.9
$K = 640$	73.8	24.0	–	–
$K = 652$	–	–	78.2	22.9

cannot be combined—SELF values should be large (disturb the valid user rarely) while the OTHER values should be small (detect an impostor quickly). It appears here that the time-to-false-alarm (SELF) values for clustering methods are occasionally better than those produced by the base-line system, but the results turn out not to be significantly different. Significant differences do, however, occur in the time-to-true-alarm (OTHER) values. Here the base-line is better than the greedy system for the Calgary users and both base-line and greedy are better than the K-centers method on both user populations. Conveniently, these results are in agreement with the accuracy results given above, though TTA is arguably the more practically useful measurement.

We note, in passing, that the clusters constructed by the greedy clustering algorithm make intuitive sense, in terms of the actions being performed by the underlying sequences. For example, we have identified clusters that correspond to "programming", "paper writing", "reading email", and "navigating directories". An example of such an "intuitive" cluster is shown in figure 8.

5.2. HMM model scaling: Number of hidden states

We turn now to an examination of scaling issues for the HMM based sensor systems. An open question in the use of HMMs for modeling is the choice of K, the number of hidden

```
**EOF** **SOF** elm findall finger <1> elm cd <1> ls
cd <1> cat <1> finger <1> cd cd <1> ls
<1> elm nn findall finger <1> elm cd <1> ls
**EOF** **SOF** elm findall finger <1> finger <1> chfn finger
**EOF** **SOF** elm findall finger <1> exit **EOF** **SOF** **EOF**
fg cd elm findall finger <1> talk <2> findall **EOF**
```

Figure 8. An example cluster produced by the greedy clustering algorithm from Purdue User 7's data. Each line is a single subsequence of commands and flags. The first line shown is the center point of the cluster. The symbols `**SOF**` and `**EOF**` denote the start and end of shell sessions, respectively.

states. When the domain suggests physical interpretations for hidden states—as in fault monitoring where hidden states correspond to failure modes—it may be possible to select K a priori. When K is not so conveniently available, however, we can employ an empirical analysis to discover an appropriate value. To examine the impact of K on sensor performance, we constructed models with $K \in \{1, 2, 5, 10, 15, 50, 100\}$ and tested them under the same conditions used for $K = 30$. The case $K = 1$ is a degenerate form of an HMM equivalent to frequency estimation of the alphabet symbols assuming temporal independence. Effectively, the data are considered to have been generated by a multinomial process with $|\Sigma|$ elements drawn according to the distribution B (the output symbol generation distribution).

Statistical summaries of HMM performance at different values of K are given in Tables 5 and 6. It is apparent from these results that the number of hidden states present in the user profile has little impact on recognition of the profiled user. In fact, the values given in the SELF columns are statistically indistinguishable. Model complexity seems to have effect only on true detection (OTHER columns), where we find a curve that peaks between $K = 2$ and $K = 15$. The absolute optimum for the Calgary data, at $K = 5$, turns out to be statistically significant but the variance in optimal model complexity per user is high, so it would be unwarranted to claim a single best value of K for all users. We have even found cases in which the single-state HMM is the best model of particular,

Table 5. Mean accuracy values for the HMM sensor across settings of K at an acceptable false alarm rate of $r = 0.5\%$. The setting $K = 30$ corresponds to the "base-line system performance" discussed in Section 4.5.

	Purdue			Calgary		
	SELF	OTHER	TOTAL	SELF	OTHER	TOTAL
$K = 1$	72.1	68.2	68.6	69.6	78.0	77.9
$K = 2$	68.6	74.5	73.7	70.2	85.3	85.1
$K = 5$	68.5	74.2	73.5	70.0	85.8	85.7
$K = 10$	69.3	74.5	73.9	69.4	85.3	85.1
$K = 15$	69.7	73.7	73.2	69.0	85.3	85.1
$K = 30$	71.1	73.4	73.1	69.5	84.2	84.1
$K = 50$	72.7	72.7	72.7	68.9	82.9	82.7
$K = 100$	72.3	71.4	71.5	70.9	81.8	81.7

Table 6. Mean TTA values for the HMM sensor across settings of K at an acceptable false alarm rate of $r = 0.5\%$. Values are reported in mean token counts. The setting $K = 30$ corresponds to the "base-line system performance" discussed in Section 4.5.

	Purdue		Calgary	
	SELF	OTHER	SELF	OTHER
$K = 1$	79.4	28.2	84.6	25.1
$K = 2$	79.9	27.4	84.2	15.8
$K = 5$	82.5	26.8	85.2	15.3
$K = 10$	83.8	26.1	83.0	16.0
$K = 15$	87.2	26.4	81.5	15.7
$K = 30$	83.8	25.7	82.3	16.5
$K = 50$	82.2	25.9	78.5	16.9
$K = 100$	85.0	26.1	83.0	19.0

low-behavioral-complexity users (Lane, 1999). A better strategy would be to adaptively seek K on a per-user basis—a tack that we are pursuing in current research.

At first glance, these results are somewhat disheartening. If even a simple multinomial model is capable of representing the profiled user as well as a complex, 100-state model, why should we bother to strive for a time-series model at all? Note, however, that the $K = 1$ model achieves its true accept performance at a significant cost in ability to distinguish impostors. Effectively, the single-state HMM has achieved coverage of the profiled user only through overgeneralization. The very large models also yield poorer performance than the midrange models. This is possibly unsurprising because the 100-state model contains far more free parameters to train than there are instances in the training data.

The most promising result of the HMM model scaling investigation is that relatively small HMMs can be very effective user models—more so than either of the "base" user models examined in Sections 4.4 and 4.5. We attribute this to a successful tradeoff between complexity of the model necessary to capture user behaviors in a discriminating fashion and simplicity of the model, which constrains the parameter space.

5.3. Summary of model scaling issues

Bringing together the results that we have explored in this section, we can make the following remarks about model scaling issues for the techniques we have examined within the anomaly detection domain:

- The two clustering methods employed with the instance-based learner exhibit different strengths in this domain. The K-centers method is capable of achieving much higher levels of model compression than is the greedy clustering method, but the former yields poorer impostor detection performance over the compression range that the two techniques have in common. The two methods are equivalent to each other and to the base-line system on this data with respect to true acceptance performance.

- The hidden Markov user models are also insensitive to scaling with respect to true acceptance abilities. Only when examining ability to differentiate impostors does it become clear that both very small models (single hidden state) and very large (50 or 100 hidden states) are inferior to more moderate models. There is a large variance of appropriate model size with respect to user, but it appears that a substantial scaling down of model size from the "base-line" model can be achieved with no performance penalties.

The common thread in these observations is that recognition of the profiled user is not significantly penalized by model scaling in either class of learning system; effects of model scaling only become apparent when detecting the presence of impostors. We conceive of two hypotheses to explain this phenomenon. The first notes that the decision rule employed is the same in both methods (see Section 3.2.5) and allows us to control only the false alarm rate. Coverage of the profiled user is maximized, even in ill-fitting models. Variation in the false accept rate is a side-effect of forcing poor models to fit the profiled user.

The second hypothesis notes that not only are true acceptance rates independent of model scale, they are also indistinguishable for the two model classes examined here. It is possible that the true acceptance rates seen here are the best achievable under a general class of models that subsume those discussed in this article—perhaps the general class of Markovian or finite-state models. In this case, all of these models are capable of representing only a subset of the profiled user's behaviors, so complete accuracy cannot be achieved. All model scales can reach that accuracy plateau but not exceed it, though there is some variability in detection rates related to the degree of overgeneralization necessary to achieve the true acceptance plateau. Perhaps another model class could achieve a higher accuracy plateau, or perhaps there is an underlying stochasticity in the data which would frustrate all possible model classes. There is some support for the hypothesis that the user identification problem is uniformly difficult for a large class of algorithms. A number of other researchers have examined this problem or close variants and found comparable results under a wide variety of methods including neural networks (Ryan, Lin, & Miikkulainen, 1997), Markov chains (Davison & Hirsh, 1998; DuMouchel & Schonlau, 1998), multi-step Markov chains, Lempel-Ziv compression, and command "unexpectedness" (Theus & Schonlau, 1998). An unpublished comprehensive study, which examined many of these in a unified framework, found them all to be competitive and none to be strictly dominant in this domain (Schonlau, 2000). We think it likely, therefore, that substantial further progress in this domain will require an entirely different class of user models.

6. Conclusions and future work

This work has demonstrated two classes of methods for approaching the human modeling aspect of anomaly detection. Both models are statistically indistinguishable on the task of identifying the true user, though HMMs hold a small but significant advantage on the task of discriminating impostors. This advantage, which is especially important when employing such a learner as a focus of attention sensor in a multi-tier decision system, may be offset in the same context by the higher computational burden of HMMs.

We examined methods for scaling the user model by removing instances from the IBL profile or by changing the number of hidden states in the HMM profile. We found that for both model classes, scaling has no significant effect on acceptance of the profiled user. It does, however, have a substantial impact on the ability to recognize impostors. Taken with the previous result, that the two model classes are indistinguishable with respect to the profiled user, we have the most important single result of this article: *profiling accuracy on the valid user is invariant under all experimental conditions we have examined*. We hypothesize two possible explanations for this result: that some common property of the two learning methods we employ (perhaps the decision rule) limits them to similar performances on this task, or that performance is limited for an entire general category of learning methods (perhaps all Markovian models). Examination of these hypotheses is one of the primary directions of future work. Furthermore, since there are variations in ability to distinguish *impostors*, we intend to investigate methods for automatically selecting appropriate model classes, scales, and parameter settings with respect to that task.

We also found that the two clustering methods investigated for IBL model scaling had different strengths: the K-centers method is capable of achieving much higher compression levels than the greedy clustering method, but the latter has better performance over the compression ranges in which it does operate. We demonstrated that in many cases the HMM models can operate with substantially fewer hidden states than in the "base-line" model and realize performance improvements when doing so. This is particularly promising in light of the quadratic runtime of HMMs.

We are in the process of examining other classes of user models based on different notions of similarity (such as compressibility) or other types of temporal sequence models (such as probabilistic CFGs). Another improvement over the results given here can be realized through the application of online training. We have explored this issue, in part, in Lane and Brodley (1998) wherein we achieved promising results for online IBL methods. We are currently examining extensions of HMMs to single-step recursive updates and non-stationary domains.

The methods employed here used a fairly small proportion of the available data (command line data only, with no file names or file extensions). Employing additional data sources such as timestamps, resource consumption, and GUI events would likely improve accuracy although a feature subset search (possibly on a per-user basis) would be required to locate the appropriate feature set from the many available. Additionally, the methods presented here employ very little domain knowledge. Extending the anomaly detection sensors to employ knowledge about command semantics, for example, would also be likely to improve accuracy.

While the data used here allows us to analyze the "naïve intruder scenario", we acknowledge that it is far from representative of real attack data. Examining data drawn from real attacks would improve the generality of our results and would likely point the way toward further developments of user profiling techniques. Unfortunately, such data is often not preserved, is proprietary, or is otherwise hard to come by, but a focus for future work will be to locate publicly available intrusion data.

We believe that, in general, both the computer security and machine learning communities can benefit from further interactions. The ML community has studied many pattern

recognition techniques which could be valuable to a variety of security problems, while computer security tasks present a number of challenging issues that can motivate new research directions for the machine learning community.

Acknowledgments

We would like to thank all of the users who donated data to this project. In addition, we would like to thank the reviewers of this article and of the conference version of this article for their valuable insights and feedback. This work was carried out under NSF grant number 9733573.

Notes

1. A replay attack is one in which an attacker monitors a system and records information such as user commands. These commands can then later be "replayed" back to the system literally, possibly with the inclusion of a small number of hostile actions. Because the vast majority of the data was, in fact, originally generated by the valid user, it will appear perfectly normal to the detection sensors unless some check is made for occurrences which are *too* similar to past behavior.
2. For string evaluation alone, it suffices to use only the "forward" step, which evaluates likelihoods based on string prefixes. In combination with the "backward" step, which works with suffixes, we can extract state occupation likelihoods for individual tokens—a statistic used in learning the HMM parameters.
3. Actually, for efficiency reasons, we use an approximation to the true sequence likelihood. See Section 3.3.4.
4. The parameter W used here plays an analogous role to that of the noise-suppression filter window length employed with the IBL models, as described in Section 3.2.4.
5. Examples (usually simulated) of machine-level attack logs (such as network packet logs or system call traces) are available, but traces of *real attacks* at the *human command* level are considerably rarer. A recent call for examples of such data by the CERIAS security research center has, to date, yielded no instances of such data. Currently such data are often not stored, not because of the non-existence of such threats, but because of the lack of adequate automated analysis tools.
6. Because the Baum-Welch training algorithm for HMMs performs an iterative gradient descent, no guarantees on the convergence rate of training are available. Each iteration of the search, however, requires at least quadratic time to re-estimate model parameters.
7. Most of the clusters constructed late in the clustering process consist of instances bearing very low similarity to one another and which are visually recognizable as being quite different. Such points are clustered only by virtue of having one or two tokens in common—often by chance. These clusters may actually detract from classification accuracy by making rare exemplar instances unavailable for comparison.
8. In empirical tests, we found clustering performance to be insensitive to the choice of ϵ.

References

Aha, D., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning, 6:1*, 37–66.

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

Angulin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation, 75*, 87–106.

Aslam, J. A., & Rivest, R. L. (1990). Inferring graphs from walks. In *Proceedings of the Third Annual Workshop on Computational Learning Theory* (pp. 359–370). Rochester, NY: ACM Press.

Balasubramaniyan, J. S., Garcia-Fernandez, J. O., Isacoff, D., Spafford, E., & Zamboni, D. (1998). An architecture for intrusion detection using autonomous agents. Technical Report COAST TR 98/05, Wes Lafayette, IN: Purdue University, COAST Laboratory.

Bollobás, B., Das, G., Gunopulos, D., & Mannila, H. (1997). Time-series similarity problems and well-separated geometric sets. In *Thirteenth Annual ACM Symposium on Computational Geometry*. Rochester, NY: ACM Press.

Burl, M. C., Fayyad, U. M., Perona, P., Smyth, P., & Burl, M. P. (1994). Automating the hunt for volcanoes on Venus. In *Proceedings of the 1994 Computer Vision and Pattern Recognition Conference* (pp. 302–309). Los Alamitos, CA: IEEE Computer Society Press.

Casella, G., & Berger, R. L. (1990). *Statistical inference*. Pacific Grove, CA: Brooks/Cole.

Chenoweth, T., & Obradovic, Z. (1996). A multi-component nonlinear prediction system for the S&P 500 index *Neurocomputing, 10:3*, 275–290.

Cis (1999). NetRanger 2.2.1 user guide. Available on Cisco Documentation CD-ROM or at `http://www.cisco.com/univercd/cc/td/doc/product/iaabu/netrangr/nr221/nr221ug/index.htm`. San Jose, CA: Cisco Systems Inc.

Das, G., Gunopulos, D., & Mannila, H. (1997). Finding similar time series. In *Proceedings of The Fourth International Conference on Knowledge Discovery and Data Mining*.

Dasarathy, B. V. (1991). *Nearest neighbor (NN) norms: NN pattern classification techniques*. Los Alamitos, CA: IEEE Computer Society Press.

Davison, B. D., & Hirsh, H. (1998). Predicting sequences of user actions. In *Proceedings of the AAAI-98/ICML-98 Joint Workshop on AI Approaches to Time-Series Analysis* (pp. 5–12).

Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering, 13:2*, 222–232.

Domingos, P. (1995). Rule induction and instance-based learning: A unified approach. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada (pp. 1226–1232). San Mateo, CA: Morgan Kaufmann.

DuMouchel, W., & Schonlau, M. (1998). A fast computer intrusion detection algorithm based on hypothesis testing of command transition probabilities. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (pp. 189–193). AAAI Press.

Fawcett, T. & Provost, F. (1999). Activity monitoring: Noticing interesting changes in behavior. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*.

Fayyad, U. M., Weir, N., & Djorgovski, S. (1993). SKICAT: A machine learning system for automated cataloging of large scale sky surveys. In *Proceedings of the Tenth International Conference on Machine Learning* (pp. 112–119).

Forrest, S., Hofmeyr, S. A., Somayaji, A., & Longstaff, T. A. (1996). A sense of self for UNIX processes. In *Proceedings of 1996 IEEE Symposium on Security and Privacy*. Los Alamitos, CA: IEEE Computer Society Press.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences, 55:1*, 119–139.

Fukunaga, K. (1990). *Statistical pattern recognition* (2nd edn.). San Diego, CA: Academic Press.

Gordon, S. (1996). Current computer virus threats, countermeasures, and strategic solutions. White paper, McAfee Associates.

Greenberg, S. (1988). Using UNIX: Collected traces of 168 users. Technical Report 88/333/45, Alberta, Canada: University of Calgary, Department of Computer Science. Includes tar-format cartridge tape.

Heberlein, L. T., Dias, G. V., Levitt, K. N., Mukherjee, B., Wood, J., & Wolber, D. (1990). A network security monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy* (pp. 296–304).

ISS (2000). RealSecure product datasheet. Available at `http://www.iss.net/customer_care/resource_center/product_lit/`. Atlanta, GA: Internet Security Systems.

Juang, B.-H. (1984). On the hidden Markov model and dynamic time warping for speech recognition—A unified view. *AT&T Bell Laboratories Technical Journal, 63:7*, 1213–1243.

Kumar, S., & Spafford, E. (1994). An application of pattern matching in intrusion detection. Technical Report CSD-TR-94-013, West Lafayette, IN: Purdue University, Computer Science.

Laird, P., & Saul, R. (1994). Discrete sequence prediction and its applications. *Machine Learning, 15:1*, 43–68.

Lane, T. (1998). Filtering techniques for rapid user classification. WS-98-07, Menlo Park, CA: AAAI Press.

Lane, T. (1999). Hidden markov models for human/computer interface modeling. In *Proceedings of the IJCAI-99 Workshop on Learning About Users (Sixteenth International Joint Conference on Artificial Intelligence)* (pp. 35–44).

Lane, T. (2000). Machine Learning Techniques for the Computer Security Domain of Anomaly Detection. Ph.D. thesis, W. Lafayette, IN: Purdue University, Electrical and Computer Engineering.

Lane, T., & Brodley, C. E. (1997a). An application of machine learning to anomaly detection. In *Proceedings of the Twentieth National Information Systems Security Conference* (Vol 1, pp. 366–380). Gaithersburg, MD: The National Institute of Standards and Technology and the National Computer Security Center, National Institute of Standards and Technology.

Lane, T., & Brodley, C. E. (1997b). Detecting the abnormal: Machine learning in computer security. Technical Report TR-ECE 97-1, W. Lafayette, IN: Purdue University, Electrical and Computer Engineering.

Lane, T., & Brodley, C. E. (1997c). Sequence matching and learning in anomaly detection for computer security. In *Proceedings of AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management (Fourteenth National Conference on Artificial Intelligence)* (pp. 43–49).

Lane, T., & Brodley, C. E. (1998). Approaches to online learning and concept drift for user identification in computer security. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (pp. 259–263). Menlo Park, CA: AAAI Press.

Lane, T., & Brodley, C. E. (1999). Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security, 2:3*, 295–331.

Lee, W., Stolfo, S., & Chan, P. (1997). Learning patterns from UNIX process execution traces for intrusion detection. In *Proceedings of AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management (Fourteenth National Conference on Artificial Intelligence)* (pp. 50–56).

Lee, W., Stolfo, S. J., & Mok, K. W. (1998). Mining audit data to build intrusion detection models. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (pp. 66–72). Menlo Park, CA: AAAI Press.

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

Moon, T. K. (1996, November). The expectation-maximization algorithm. *IEEE Signal Processing Magazine,* 47–59.

Norton, S. W. (1994). Learning to recognize promoter sequences in E. coli by modelling uncertainty in the training data. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA (pp. 657–663).

Oppenheim, A., & Schafer, R. (1989). *Discrete-time signal processing*. *Signal processing*. Englewood Cliffs, NJ: Prentice Hall.

Orwant, J. (1995). Heterogeneous learning in the Doppelgänger user modeling system. *User Modeling and User-Adapted Interaction, 4:2*, 107–130.

Pfleeger, C. P. (1997). *Security in computing* (2nd edn.). Upper Saddle River, NJ: Prentice Hall PTR.

Porras, P., & Neumann, P. (1997). EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the Twentieth National Information Systems Security Conference* (pp. 353–365). Gaithersburg, MD: The National Institute of Standards and Technology and the National Computer Security Center, National Institute of Standards and Technology.

Power, R. (1998). *Current and future danger: A CSI primer on computer crime & information warfare*. San Francisco, CA: Computer Security Institute.

Provost, F., & Fawcett, T. (1998). Robust classification systems for imprecise environments. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77:2*.

Rabiner, L., & Juang, B. H. (1993). *Fundamentals of speech recognition*. Englewood Cliffs, NJ: Prentice Hall.

Rivest, R. L., & Schapire, R. E. (1989). Inference of finite automata using homing sequences. In *Proceedings of the Twenty First Annual ACM Symposium on Theoretical Computing* (pp. 411–420).

Ryan, J., Lin, M.-J., & Miikkulainen, R. (1997). Intrusion detection with neural networks. In *Proceedings of AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management* (pp. 72–77). AAAI Press.

Salzberg, S. (1991). A nearest hyperrectangular learning method. *Machine Learning, 6:3*, 251–276.

Salzberg, S. (1995). Locating protein coding regions in human DNA using a decision tree algorithm. *Journal of Computational Biology, 2:3*, 473–485.

Schaffer, C. (1994). Cross-validation, stacking, and bi-level methods for stacking: Meta-methods for classification learning. In P. Cheeseman, & W. Oldford (Eds.), *Selecting models from data: Artificial intelligence and Statistics IV*. New York: Springer-Verlag.

Schonlau, M. (2000). Personal communication.

Sheskin, D. J. (1997). *Handbook of parametric and nonparametric statistical procedures*. Boca Raton, FL: CRC Press.

Shyu, C. R., Kak, A. C., Brodley, C. E., & Broderick, L. S. (1999). Testing for human perceptual categories in a physician-in-the-loop CBIR system for medical imagery. In *Proc. IEEE Workshop of Content-Based Access of Image and Video Databases*, Fort Collins, CO.

Smaha, S. E. (1988). Haystack: An intrusion detection system. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference* (pp. 37–44).

Smyth, P. (1994a). Hidden Markov monitoring for fault detection in dynamic systems. *Pattern Recognition, 27:1*, 149–164.

Smyth, P. (1994b). Markov monitoring with unknown states. *IEEE Journal on Selected Areas in Communications, special issue on Intelligent Signal Processing for Communications, 12:9*, 1600–1612.

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

Stough, T., & Brodley, C. E. (1997). Image feature reduction through spoiling: Its application to multiple matched filters for focus of attention. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*.

Theus, M., & Schonlau, M. (1998). Intrusion detection based on structural zeroes. *Statistical Computing & Graphics Newsletter, 9:1*, 12–17.

Wespi, A., Darcier, M., & Debar, H. (1999). Intrusion detection using variable-length audit trail patterns. Technical Report RZ 3164 (# 93210), Zurich, Switzerland: IBM Research.

Wilson, D. R., & Martinez, T. R. (2000). Reduction techniques for exemplar-based learning algorithms. *Machine Learning, 38:3*, 257–268.

Yoshida, K., & Motoda, H. (1996). Automated user modeling for intelligent interface. *International Journal of Human-Computer Interaction, 8:3*, 237–258.

Received July 11, 2001
Accepted March 14, 2002
Final manuscript November 4, 2002