Machine Learning Techniques for the Domain of Anomaly Detection for Computer Security

Terran Lane
Purdue University
Department of Electrical and Computer Engineering and the COAST Laboratory

July 16, 1998

Keywords: Concept Drift, Lifelong Learning, Sequence Learning, Hidden Markov Models, Classification, User Recognition, Computer Security, Anomaly Detection.

Contents

List of Figures

List of Tables

Abstract

In this proposal, we examine the machine learning issues raised by the domain of anomaly detection for computer security. The anomaly detection task is to recognize the presence of an unusual (and potentially hazardous) state within the behaviors or activities of a computer user, system, or network with respect to some model of 'normal' behavior which may be either hard-coded or learned from observation. We focus, here, on learning models of normalcy at the user behavioral level, as observed through command line data. An anomaly detection agent faces many learning problems including learning from streams of temporal data, learning from instances of a single class, and adaptation to a dynamically changing concept. In addition, the domain is complicated by considerations of the trusted insider problem (recognizing the difference between innocuous and malicious behavior changes on the part of a trusted user) and the hostile training problem (avoiding learning the behavioral patterns of a hostile user who is attempting to deceive the agent). We propose an architecture for a learning anomaly detection agent based on a hierarchical model of user behaviors. The leaf level of the hierarchy models the temporal structure of user observations, while higher levels express interrelations between descendant structures. We describe approaches to fabrication of such models, employing instance based learning models and hidden Markov models as the fundamental behavioral modeling units. Approaches to the trusted insider and hostile training problems are described in terms of the hierarchical behavior model. Finally, we present empirical results for a prototype anomaly detection system which employs an instance based learning model in a single level model.

Chapter 1

Introduction

1.1 Motivation and Background

Consider your own activities on a 'standard' day—you may interact with dozens of people in person or remotely, read, write, drive, work, cook, plan, play. You are engaged in a myriad of tasks of varying degrees of importance and requiring varying degrees of attention. But these tasks are almost certainly different, in detail if not in kind, than the daily tasks you were engaged in a year ago, five years ago, ten years ago. You have adapted to changing circumstances and demands. In part, your current actions and responses reflect the unique demands of the current situation, but to a large degree they are also shaped out of previously learned behaviors and skills. This ability to act by recycling previously gained knowledge when possible while acquiring new knowledge or skills when necessary is one of the greatest strengths of human intelligence and is one of the major contributing factors to human adaptability.

A desirable goal is to replicate a human-like degree of adaptability for use in (semi-)autonomous agents. Such agent programs, if they are expected to endure and function for extended periods in environments approaching the complexity of the natural world (or even an advanced synthetic environment such as the Internet), must be able to react to continually changing environments. Behaviors must be acquired and remembered while useful and disregarded when no longer relevant. As tasks change, an agent is presented with new challenges and must be able to recognize similarities to previously encountered situations. In changing circumstances, it must be able to apply familiar problem solving techniques as well as to develop novel solutions when previous skills are inadequate. Furthermore, to decide when and how to apply retained knowledge and when to search for new, the agent must be able to recognize that some facet of the situation has changed and in what way.

There are a number of ways in which such dynamic tasks differ from the classical static machine learning domains, but we devote our attention to two important problems. The first is representation and comparison of temporal sequences of discrete events for learning, and the second is detection of and adaptation to *concept drift*, which we define here to be the change, with time, of the underlying concept to be learned for a given domain.

A great deal of machine learning research over the past two decades has been devoted to the problem of acquiring behaviors or knowledges suitable to a static single domain. Although the assumption of conceptual stasis is often a simplification of real environments, it's an extremely useful assumption and is often perfectly adequate to the task at hand. In general, however, a great many tasks *don't* exhibit this static nature. Now that time invariant learning tasks are becoming well understood, there is growing research interest directed to dynamic and adaptable systems. Studies have examined issues such as concept drift [55], learning bias from multiple tasks [7], continual learning [53, 57], multitask learning [9], knowledge transfer between tasks [46, 44], and lifelong learning [69]. While progress has been made, this branch of study still contains many unresolved issues. Methods are needed to reliably detect when the underlying concept being modeled has changed since training began and to adapt the current domain model to the new conditions. It may even be necessary to alter the learning *bias* (the heuristic that guides the learning algorithm in its search for a domain model) in order to adapt the current model to the new concept.

One domain in which continual learning or other forms of adaptive modeling show great promise is human behavioral modeling. While the general field of human behavior is enormously complex, the subset of behaviors exhibited by humans in interaction with a computer is relatively simple and more easily examined, yet still provides a rich domain for investigation. In particular, we have found that the *anomaly detection* domain, widely studied in the computer and information security community (see, for example, [16, 39, 28]) is a fruitful domain for the examination of issues in machine learning for dynamic systems. The general goal of anomaly detection is to model the state of a computer system, network, or user and to detect later misuse in terms of deviations from the known patterns. In this work, we focus on a personal anomaly detection agent, which assists a user by learning that user's behaviors in order to help protect his or her account or system from unauthorized access. An adaptive learning model is a promising approach to profiling user behavior for such anomaly detection systems [31, 30, 32]. This domain is a particularly interesting learning context as it presents a number of difficulties to an adaptive learning system:

- The system must be both adaptable to change on the part of a valid user, yet resistant to intruders masquerading as authorized users or attempting to train the system away from the valid user's profile.
- For reasons of privacy and practicality, it is often necessary to train such a system with data from only a single user and we thus have the task of inducing a concept from examples of only a single class.
- The anomaly detection domain presents us with a stream of discrete events (commands, system calls, user interface events, etc) from which we wish to induce a concept. This task of learning from temporal sequence data has its own issues that bear investigation such as learning on non-metric spaces and learning relations within feature vectors.

In this research proposal, we will examine issues surrounding the task of continual learning for autonomous agents as applied to the anomaly detection for computer security domain. We present a detailed description of the domain and the machine learning issues that it raises, and we outline the scope of the proposed research. In the following chapters, we examine related work in the machine learning and computer security communities, present a preliminary implementation and current experimental results, and finally describe the proposed research.

1.2 Goals of this Research

We divide the proposed research into two broad branches: issues involved with continual learning and concept drift, and issues involved with representation of and learning from time sequence data. We briefly describe each of these branches, here, and its relation to the anomaly detection domain.

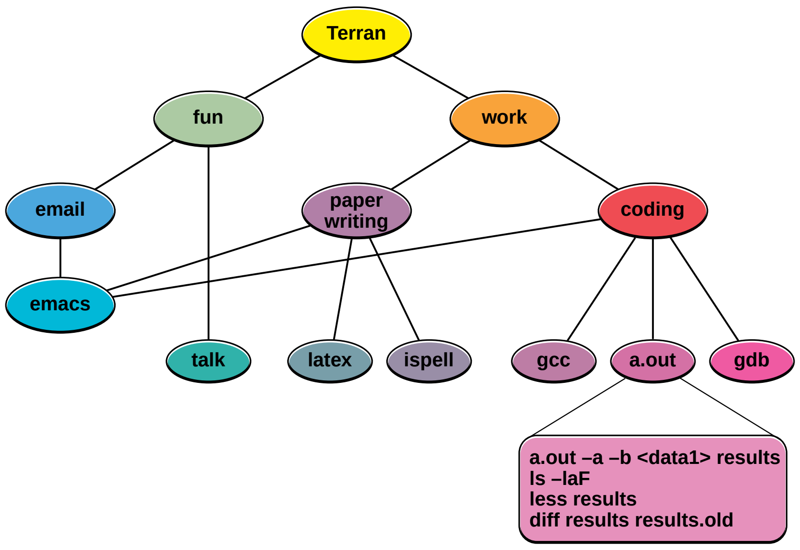


Figure 1.1: An example behavioral hierarchy for human/computer interactions.

1.2.1 A Note on Terminology

Although the work presented in this proposal can be extended to analysis of any stream of discrete events such as graphical user interface events, command names, or system calls, our implementation focuses on learning from UNIX shell command traces. Therefore, use of the terms 'command', 'interface event', or 'atomic event' in the general discussions that follow should be taken to refer to any discrete event.

1.2.2 A Hierarchical Model of Behavior Learning

Our approach to human behavioral learning is based on a causal model of human/computer interaction. The hypothesis is that a user approaches a computer system with a goal in mind and that interactions with the machine are driven by that goal. The human's actions and responses are driven by the context of the goal, while the computer's actions and responses are driven by its internal state. Furthermore, we believe that the hierarchical model of problem solving embodied in most modern computer systems will evidence itself in the user's behaviors. Thus, we are interested in inducing hierarchies of behavioral patterns, where each level of the hierarchy is modeled as temporal sequences of discrete events. An example of such a hierarchy is displayed in Figure 1.1. The lowest level of the hierarchy (leaf nodes) is comprised solely of atomic user events, while higher levels are comprised of conglomerations of lower level events representing tasks or subtasks. The root of the tree should represent the profiled user. In this framework, detection of concept drift can be modeled as detection of deviations from 'normalcy' or expected behavior. Adaptation to drift depends on the underlying model used for temporal grouping. Deviations from normalcy can occur at any level of the hierarchy and agent reactions depend on the hierarchy level in question. Deviations at the leaf level represent usage of previously unseen commands while deviations at higher levels indicate introduction of new subtasks or even goals. Anomalies at the lower levels of the tree probably indicate learning on the part of the user and should be used to adapt the current model of the user. Anomalies at the higher levels, on the other hand, may indicate an intruder or abusive actions and should be flagged by the system as suspicious.

1.2.3 Sequence Learning

A key component of the hierarchical continual learning model presented above is the ability to learn patterns of events in temporal sequences. There are two major ways in which the sequence learning problem differs from many previously examined learning tasks. Temporal learning has been studied for time series data, but almost exclusively for numerical time series in which the existence of a full ordering and a distance metric allow the application of many powerful mathematical techniques. Time series of discrete elements, however, do not inherently possess such distance properties so techniques such as spectral analysis [42], clustering [21], or neural networks for temporal prediction [12] are not directly applicable.

Discrete, non-metric spaces have been studied, but mostly for the atemporal case, in which an instance is considered to be a feature vector in which element ordering is not significant. Decision trees [47], summary statistics [34], rule learners [48], and even k-nearest neighbor classifiers [2] have been used for learning on discrete valued spaces but all of these methods consider each feature independently. That is, the contribution of a single feature to the final classification depends only on its value and not on its position within the sequence, its relation to the preceding and succeeding features, etc. Thus, a straightforward application of such techniques (in which each time step within a fixed length sequence is considered to be a single feature element) will ignore the information contained within the temporal relations among elements. Such methods also typically depend on multi-class training data.

We propose to investigate the structure of the discrete valued temporal sequence learning problem. In particular, we are interested in the use of both an *instance based learning* (IBL) model (similar to nearest neighbor based classification) and *hidden Markov models* (HMM's) as representations for this problem. While examining these algorithms, we must keep in mind that the domain demands low space and time overhead for learning and classification. There is potentially a huge volume of data available, but the anomaly detection system must not be obtrusive to normal system usage. Both IBL and HMM learning methods store dictionaries of behavior exemplars[1] and classification requires time linear in the size of the dictionary. Thus, we require methods for constraining the storage space required for each method.

1.2.4 Continual Learning and Concept Drift

A personal anomaly detection agent must be capable of adapting to its user's behavioral changes, as observed through new observed behaviors. At the same time, it must avoid acquiring new behaviors that result from an intruder or from hostile activities. In the worst case, the anomaly detection system must resist the actions of a hostile user with perfect knowledge of the detector's learned model, who attempts to subvert its learning strategy with carefully planned and seemingly innocuous behaviors.

We describe the required adaptive abilities of an anomaly detection agent in terms of the *continual learning* paradigm proposed by Ring [52, 53]. Continual learning, as described by Ring, is the process of behavioral adaptation engaged in by a learning agent embedded in a potentially changing, partially observable environment for which a single skill or knowledge set is insufficient. There are two basic sub-tasks that must be handled by a continual learning system to successfully adapt to changing environments. Initially, the agent must detect that conditions have changed and that currently held knowledge is no longer adequate. Once the agent is aware that change has occurred, it must adapt its internal model to the current situation. This may be accomplished by immediately discarding the current model and inducing a new

[1]In the hidden Markov model context, the exemplars are individual models that represent classes of user behaviors.

one, or by altering the current model to fit new circumstances. Note that the two phases may not be separated within the learning algorithm; techniques that automatically discard training instances by age, for example, integrate the two phases with an assumption of drift. In general, the process of adapting current models to new information has been studied under the rubrics of multitask learning [9], knowledge transfer between tasks [46, 44], and lifelong learning [69].

Chapter 2

Issues and Related Work

In this chapter we will discuss the goals of the anomaly detection domain, related background work, and the issues raised by the proposed research. Although we make an effort to divide the issues into those most nearly learning related and those most nearly security related, we note that the nature of the domain is such that it can be difficult to completely separate the two.

2.1 Requirements for an Adaptive Anomaly Detection Agent

Before examining the anomaly detection problem in detail, we discuss the overall performance goals of an adaptive anomaly detection system. When examining machine learning algorithms for anomaly detection, we must keep in mind several practical requirements imposed by the domain and intended use. Specifically, the purpose of any security system is to enhance the users' ability to accomplish their desired tasks. In the context of anomaly detection, this enhancement is increased confidence on the parts of individual in the privacy and confidentiality of personal systems or accounts. From a global perspective, strong anomaly detection systems increase confidence in authenticity (the belief that actions originating with a particular account are actually associated with the owner of that account) and increase assurance that shared system resources and data are being used properly.

In this section, we review personal versus global definitions of 'anomaly', requirements for system accuracy, and space and time resource issues.

2.1.1 The Scope of 'Anomalous'

The class of anomalous activities includes a wide variety of different activities. Intuitively, what we are trying to detect is 'hazardous', 'hostile', or 'abusive' activities. Unfortunately, these concepts are quite loosely defined and, in the final analysis, depend on human desires and goals. With respect to a single user, employing an anomaly detection agent to monitor a single personal computer system or account, this notion is close to 'intrusive'. All actions undertaken by the system owner are, by definition, in accord with that person's desires and the (implicitly defined) security policy. The system owner *is* likely to be interested in actions that seem to originate with an outsider.

When the anomaly detector is employed as an assistant to a network security officer, on the other hand, the notion of 'abusive' is somewhat broader than just 'intrusive'. In this context, there is a security policy that defines acceptable system uses. While we still wish to detect illicit system use by intruders, we are also interested in abusive activities by authorized users. One legitimate user may be accessing another's account to gain unfair advantage, for example. Or, perhaps, a user is disseminating controlled information to untrusted parties or to competitors.

2.1.2 Accuracy and Error Rates

To increase general user confidence of privacy and authenticity, the anomaly detection agent must be accurate at detecting intrusions and hostile activities. At the same time, though, it must not raise false alarms too often; users and system administrators will quickly learn to ignore the 'security agent that cried wolf', if it often flags innocuous behavior as hostile. We thus have two classes of errors with which we are concerned: the *false accept* rate (rate of accepting hostile behaviors as anomalies) and the *false alarm* rate (rate of incorrectly rejecting the valid user). The converse measures are *true detect* and *true accept* rates. While we wish both error rates to be as low as possible, there is inevitably a tradeoff between complimentary error classes, so a decision must be made as to what error rates are considered 'acceptable'. The tradeoff between acceptable levels for the two error rates is dictated by security policy on a site by site basis. In light of these demands, we require that the error rate tradeoff be governed by a user-selectable parameter.

2.1.3 Resource Consumption

Beyond merely being accurate, a security system must not become obtrusive through time or space resource consumption. A perfectly accurate anomaly detector will still be useless if it consumes *all* available resources. In practical terms, the anomaly detector must not introduce unreasonable CPU loads or storage requirements. Because there is usually a tradeoff between resource consumption and model accuracy, the definition of 'reasonable' resource allocation should also be user selectable.

2.2 Machine Learning Issues

We divide the machine learning issues into continual learning and temporal sequence induction. Continual learning and the related fields of concept drift, lifelong learning, and knowledge transfer are concerned with dealing with learning in a changing environment where previously acquired knowledge can be useful to future tasks. The anomaly detection domain is such an environment, as the user being monitored changes tasks, knowledge, and behavior over time. By temporal sequence induction, we refer to the process of learning from temporally distributed sequences of *discrete*, *unordered* elements, such as occur when examining audit trails or command traces.

Before discussing the literature and its relations to our own work, we introduce the following framework for describing machine learning algorithms in general: Learning algorithms consist of two major parts: the *hypothesis language* (the language used to represent learnable concepts as partitions of a feature space) and the *bias*, [38], (the heuristic used to search the hypothesis space for appropriate generalizations). The hypothesis language constrains the possible concepts learnable by the algorithm and should general enough to cover the complete space of concepts of interest. For many commonly employed learning systems, the hypothesis language is actually capable of representing any possible function on the feature space. Although significant increases in learning rate can be realized if the hypothesis language is restricted, it may be difficult or impossible to guarantee that interesting concepts are still expressible under the restricted language. It is, therefore, generally the bias that defines the power of the learning algorithm for a particular task.

2.2.1 Concept Drift and Continual Learning

In a domain, such as the anomaly detection domain, where the target concept is subject to change with time, it is not possible to locate a single, static, optimal hypothesis. This property was named *concept drift* by Schlimmer in [55] where he studied the capacities of two learning methods to adapt to concept changes in a synthetic domain. In his experiments, the drift could be modeled precisely and controlled by the experimenter so that analysis of learner effectiveness was straightforward. Schlimmer employed radical concept drifts, in which the entire concept was instantaneously replaced with an unrelated concept. Since Schlimmer's work, a number of researchers have examined the problem of dealing with concept drift in different domains. We are interested in domains in which concept drift is not radical. That is, a majority of the concept (some user behaviors, in our domain) remains unchanged and should be preserved while a segment of it requires update. The questions of recognizing that drift has occurred, locating the appropriate sections of the model to adapt, and adapting the model as efficiently as possible has been studied under the titles continual learning, knowledge transfer, lifelong learning, and multitask learning.

The term *continual learning* was coined by Ring to describe the learning process employed by autonomous, long-lived agents in changing environments. Specifically, in [53] he gives the following seven criteria for a continual learner:

- [A continual learner] Is an autonomous agent. It senses, takes actions, and responds to the rewards in its environment.
- Can learn context-dependent tasks, where previous senses can affect future actions.
- Learns behaviors and skills while solving its tasks.
- Learns incrementally. There is no fixed training set; learning occurs at every time step; and the skills the agent learns now can be used later.
- Learns hierarchically. Skills it learns now can be built upon and modified later.
- Is a black box. The internals of the agent need not be understood or manipulated. All of the agent's behaviors are developed through *training*, not through direct manipulation. Its only interface to the world is through its senses, actions, and rewards.

- Has no ultimate, final task. What the agent learns now may or may not be useful later, depending on what tasks come next.

Following these guidelines, he developed a reinforcement oriented maze navigation agent around his Temporal Transition Hierarchy learning algorithm [52]. Although the requirements for the anomaly detection agent that we present here differ in some key aspects from Ring's definition, we feel that the overall classification of an anomaly detection agent as a continual learner is appropriate. In particular, the anomaly detection system has the following requirements:

- It should be an autonomous agent. We conceive of this system as a software assistant that monitors a single user's account, reporting to the account owner or security officer when it detects abuses. Security measures should be unobtrusive, however, so the agent should have as little direct interaction (other than passive data collection) with the user as possible. It must, therefore, learn and act independently and with very little behavioral feedback. It has no expected termination point, being active for the entire lifetime of the user's account.

- It must learn from examples of only a single class, recognizing events that are 'suspiciously' different from that class. Because it's impractically difficult to characterize the space of abusive, anomalous, or suspicious actions in an *a priori* fashion, the space can only be described by example (thus, the need for a learning system in the first place). But by sampling from multiple users, we cannot be sure of obtaining enough training data to fully describe such a space; there always remains the possibility that an intruder will possess a behavioral pattern different from any previously observed. Furthermore, privacy issues often limit the type and amount of data available from users other than the account owner, while efficiency considerations dictate that the model should be compact and not reserve space characterizing parts of the space not used in the primary task of recognizing the target user.

- It should be able to recognize events within patterns and patterns within larger contexts. Individually innocuous events (a file copy, for example) may be seen to be suspicious within larger contexts (if the file being copied is the system password file). This argument can also be extended to higher levels of behavioral patterns; file system browsing may be suspicious in the context of email reading, and one user's normal behaviors may be another's attack pattern.

- It should constantly acquire new skills. Although there is only a single task to be solved — recognition of the valid user (or, conversely, recognition of anomalous events) — multiple skills can contribute to the task. If a recognizer for a behavioral pattern is taken to be a skill, then the agent's job is to continuously acquire new skills and learn the appropriate contexts for their applications.

- The agent must constantly both learn and apply its knowledge. Because the user's tasks and behaviors change with time, no single training set can be sufficient to cover the necessary behaviors.

- It should be capable of detecting a hierarchy of behavioral patterns. We hypothesize that the hierarchical formulation of problem solving will be manifested in the user's behaviors, and that this hierarchical structure can be induced from the user's input data. Thus, 'higher level' behavioral recognizers (task or goal recognizers, for example) can make use of 'lower level' recognizers developed previously.

- The agent should learn and function independently of human direction. Thus, its internal model needn't to be manipulable by an outside observer. This has the dual benefit of relieving a human from the task of constantly updating a rule base (the goal of an adaptive learning system in the first place) and obscuring the model's details from a malicious user who might wish to emulate or modify them.

- It must constantly and unceasingly adapt and acquire new skills. The anomaly detection agent has no 'final goal'. Its lifetime is the lifetime of the user's account. While its goal is constant — monitor the account for unauthorized use or abuse — the tasks and skills necessary to accomplish that task are mutable.

Ring's Temporal Transition Hierarchy algorithm [51, 52] builds a synchronous higher order neural network whose higher level nodes act to recognize contextual information (in terms of inputs to the network at the previous time step) and modify the connection weights between the input and output nodes.

Such a structure is inappropriate to the anomaly detection task for a number of reasons. The lack of negative instances renders the neural network training rule ineffective, and the neuronal class discrimination model (spatial regions defined by hyperplanes) is not well suited to the non-metric nature of the feature space.

Schmidhuber has examined a continual learning, reinforcement oriented domain similar to Ring's, [56, 57]. The system he develops to tackle the problem employs a generalized, Turing-equivalent programming language as a hypothesis representation language and a search engine (using Levin Search) to construct programs within this language. Programs are preserved or discarded based on an amortized reward accrued while the program is executing. The programs are permitted to be self-modifying so there is allowance for adaptive action strategies. While different in implementation, this approach has many abstract commonalities with Ring's. In particular, training is based on a reinforcement feedback signal from the environment and the agent is allowed to explore the environment as part of the learning process. Again, both of these conditions are absent from the anomaly detection domain.

The continual learning framework is conceptually similar to learning structures presented by other researchers, although the temporal continuity of their systems are often not stressed as highly. Pratt, for example, has given attention to the problem of *knowledge transfer* among different tasks [46, 44, 45]. Here the goal is to apply knowledge acquired in one learning task to another learning task in order to speed learning and possibly increase final accuracy. Pratt specifically investigates transfer among neural network structures by preserving decision hyperplanes across tasks. The arrangement of hyperplanes, as defined by the network's hidden unit weights, determines the decision structure embodied by the neural network. Literal transfer of these hyperplanes initializes a new network to the same structure as the previously learned one, but suffers from inability to adapt to differences from the previous task. To counteract this deficiency, Pratt introduces the Discriminibility Based Transfer (DBT) algorithm which selectively alters the 'inertia' (magnitude of a hidden unit's weight vector) of some hyperplanes, allowing them to more easily adapt to the new task. The selection process is based on an information metric which chooses hyperplanes that fail to effectively discriminate the new task's classes.

As with Ring's work, the neural network framework is inappropriate to the anomaly detection domain because of the non-metric nature of the data space and the lack of negative training instances. Additionally, the information metric employed by the DBT algorithm fundamentally depends on multi-class data. Finally, Pratt's entire knowledge transfer paradigm is predicated on the existence of well differentiated tasks. That is, it is possible to say that there are k different tasks and that task t_1 is defined by data $\mathbf{X}_1$, task t_2 is defined by data $\mathbf{X}_2$, and so on. While the anomaly detection domain can be formulated as consisting of multiple tasks (for example, tasks could correspond to recognizing different user behavioral patterns), such tasks are not well separated. The data for the different tasks are intermingled and there is no external indication as to which data is associated with which task (in fact, some data may well be associated with *multiple* tasks). Furthermore, there isn't even any *a priori* indication of how many tasks there are. Thus, Pratt's formulation of knowledge transfer cannot be easily applied to the anomaly detection domain.

Thrun's *lifelong learning* formulation, [69, 70, 71], frames the lifelong learning domain as the problem of learning bias from $n-1$ tasks to speed learning on the n^{th} (presumably related) task. Thrun has examined methods for transferring knowledge among learners for a memory based learner (k-nearest neighbor) and for a neural network learner. For the former, he takes two approaches: data re-representation and transfer of distance metric, while for the latter he applies the data re-representation scheme and presents an algorithm which augments the neural net training data with gradient information derived from previous models. Data re-representation is performed by employing a neural net to learn a mapping from the natural data representation to the input representation for the concept learning system, allowing the concept learner to be applied to apparently disparate tasks. Thrun also presents an analysis of the lifelong learning process that claims that the technique is effective because over multiple tasks it narrows the bias space. This narrowing has the effect of learning the invariances present among tasks, allowing a learner for a given task to specialize a generally correct framework rather than having to acquire the entire task from scratch.

Thrun's formulation of lifelong learning and the techniques he develops to implement it are inappropriate to the anomaly detection domain for many of the same reasons that Pratt's knowledge transfer techniques are. Specifically, the anomaly detection domain does not possess well defined 'tasks' that can be learned separately and data re-representation through a neural network transform is inappropriate because of the non-metric nature of the feature space.

A related problem was examined by Caruna in his *multitask learning* work

[10, 11, 9]. In this case, the learning system is presented with a goal task on which actual performance is to be measured and a number of known related tasks for which training data is available. In this case, however, the tasks are not presented sequentially but simultaneously. Caruna employs a multiple output learning model (a neural net, k nearest neighbor, and a kernel regression model for this work) to learn all tasks simultaneously. He demonstrates that for some visual recognition tasks and a medical domain task, this model can acquire the goal task more accurately than can a similar learner trained only on the goal task. This work is something of a bridge between the static, single knowledge source framework of learning and the dynamic, multiple knowledge source framework espoused by Ring. Certainly, the ability to acquire knowledge from multiple related sources is a useful one to an autonomous agent but Caruna's particular solution, based as it is on supervised learning algorithms, is not well suited to the anomaly detection task. Furthermore, like the previously discussed works, the different tasks comprising the anomaly detection domain are not easily separated *a priori*.

In [7], Baxter presented a formal analysis of multitask learning as practiced by Pratt, Thrun, Caruna, and others. His analysis is based on the hypothesis that multitask learning is effective because it improves the task-level learning bias across successive tasks. Baxter assumes that there is a true (optimal) bias which is drawn from a space of possible biases according to some distribution. Using an information theoretic model, he shows that when the individual tasks are closely related and arbitrary sampling of the task space is allowed then the bias-learning system can converge to the true bias with arbitrary accuracy. Furthermore, in these conditions, the amortized cost of learning a single task (in terms of number of examples required to learn a particular accuracy threshold) can be reduced to a bias-dependent optimal constant. Baxter's analysis applies to the case in which there is a single optimal bias across all tasks. In the temporal, concept drift case, it's possible that the bias itself is changing with time and that the optimal bias for learning task t_1 is quite different from the optimal bias for learning task t_n. Under such conditions, it is possible that there exists no single task-level bias that optimizes a fitness criterion across all tasks.

2.2.2 Temporal Sequence Learning

A second feature of the anomaly detection domain is temporal interactions between events. Human/computer interactions are generally causal and goal driven; a user commands a computer in order to achieve a particular goal and responds to the computer's reactions in such a way as to (hopefully) bring that goal closer. The process is not fully deterministic, though, as tasks may change dynamically (interrupting work on a document to answer an email message) and the same task may be accomplished via different methods on separate occasions. Finally, noise such as typos or 'mouse-o's' is likely to corrupt the signal. These two factors lead to a domain in which intra-event correlations can encode a great deal of characteristic information. Thus, we seek methods for learning characteristic temporal sequence structures in streams of discrete, unordered user observations. Current learning methods either do not handle data with temporal structure or employ a distance metric on the feature space to compare temporal sequences. In this section, we examine current approaches to temporal sequence learning and their limitations with respect to the anomaly detection domain.

A number of methods for characterizing time series data have been developed in the signal processing and pattern recognition communities [21, 42]. Among these are spectral analysis, principle component analysis, and nearest neighbor matching. Some inductive approaches, such as artificial neural networks, have also been applied to time sequence modeling, [13]. Unfortunately, such techniques depend fundamentally on the existence of a distance metric on the feature space. On an unordered, discrete space such as that defined by command strings, there is no such explicit metric. There is no sense that the string ``cat'' is 'closer to' or 'further from' the string ``vi'' than it is to ``more''. An artificial ordering can, of course, be imposed upon such a space, but without an *a priori* justification such an imposition is dubious at best.

Methods have been demonstrated for converting an unordered discrete space into a metric space without introducing an artificial ordering. For example, Aha et al. [1], employed a representation in which each discrete feature of the original data space is converted to a bit vector with one bit position for each possible value of the discrete variable. Such a representation allows standard machine learning algorithms to be applied, but expands the dimensionality of the feature space. In the case of anomaly detection, where the count of possible commands may number in the hundreds to thousands, the dimensional explosion is unacceptable because the data space quickly becomes too sparse for learning to be practical.

Alternatively, the feature space can be treated in its native form. A number of techniques have been developed for learning from discrete valued attributes. Decision trees [47, 49] and symbolic rules [48], for example, are well suited to representing decision boundaries on discrete spaces. The bias used to search for such structures, however, generally employs a greedy search that examines each feature independently of all others. This bias ignores internal relations arising from causal structures in the data generating process.

One method of circumventing this difficulty is to convert the data to an atemporal representation in which the causal structures are represented explicitly. Norton [41] and Salzberg [54], for example, 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. Salzberg's system converted each DNA sequence into a set of summary features which had been developed by domain experts. He was then able to apply a standard decision tree algorithm to the classification problem. This type of approach, while often effective, requires the time and effort of domain experts to isolate the important feature relations and develop the appropriate mappings. It is desirable to limit the necessity for human intervention as much as possible. Furthermore, reducing the dependence on hard-coded domain specific features increases the generality of the system; a system developed to analyze command traces, for example, could be easily extended to analyze system call traces if it did not have to depend on a model of command interactions. Finally, model based assumptions are often dangerous in computer security. Many attacks that penetrate computer defenses exploit deficiencies in the abstract security model or gaps between the abstract model and its concrete implementation. A system which relies too heavily on model based assumptions may risk missing such attacks.

A non model-based approach to time sequence learning is to employ *summary statistics* to reduce temporal data to atemporal. For example, a stream of command input data can be reduced to summaries such as mean usage of each command per unit time or mean and standard deviation of time between command inputs. This approach is popular for adaptive anomaly detection systems; see, for example, [26, 4, 59]. By employing statistical models for data reduction, the need for sophisticated knowledge of the target system is alleviated. A *statistical* model of the data is still required, but such models are quite general and widely applicable. The difficulty is that many summary statistics also eliminate the inter-event structures that may be critical for anomaly recognition. Many known intrusion patterns, for example, employ well known and widely used commands, such as the UNIX `finger` command, merely in novel ways [64]. Such attacks would not appear to many summary statistics because the anomaly is found not in what commands are employed, but in how.

One class of methods that are capable of modeling discrete time series data without an explicit model of system functionality is that of hidden Markov models (HMM's). HMM's have been found to be useful for many time series recognition and learning tasks such as speech recognition [50, 37], time sequence clustering [62], and fault detection [60, 61]. We hypothesize that they will also prove to be useful models for the class of temporal sequences encountered in the user behavioral modeling domain.

HMM's can be thought of as stochastically observable finite state automata with probabilistic state transitions. The hypothesis language for HMM's is the space of possible models — that is, all possible fully or partially connected graphs (representing state machines), all possible transition probabilities for those graphs (including zero probability transitions), and all possible observation processes (the stochastic process mapping the internal model state to observable variables). The learning bias is then the algorithm used to construct such models from the data. Currently, methods exist to select transition probabilities and observation process parameters from data [50] but general methods for domain-specific structure selection (i.e. choice of the size and interconnections of the state transition graph) are lacking. We present a formal description of hidden Markov models and the associated training rules in Appendix A and will discuss the structure selection issue in Chapter 4.

Hidden Markov models are an appealing learning technique for the anomaly detection domain because they can encode both deterministic and stochastic aspects of user behaviors. The state transition matrix can encode the causal aspects of behavior that are associated with accomplishing particular tasks. Because the state transitions are non-deterministic, they can also model occurrences of, for example, one task being preempted before completion in favor of another of higher priority. The stochastic output processes can help to account for noise in the stream of user observations (such as typos) and can model a single task being accomplished through multiple alternative methods.

2.2.3 Data Reduction

Instance selection is the problem of choosing a small subset of instances to be used for model construction from the pool of all available instances in order to decrease resource consumption while maintaining accuracy. The large amount of data available for user profiling dictates that some form of instance selection must be a part of any successful anomaly detection system. Until the recent advent of the field of Knowledge Mining and Data Discovery (KDD), over-abundance of data had not traditionally been a difficulty for machine learning researchers — often quite the contrary. Recently, though, a number of researchers have given attention to the instance selection problem. Lewis and Catlett [33], for example, have examined a selective sampling method that attempts to gather more information about those instances for which classification is uncertain — that is, instances for which the current classification model yields classifications with a low confidence of correctness. Their technique presupposes a learner that can interact with the data source to request more examples of the concept. It is infeasible, for most security applications, for the anomaly detector to request that a user undertake a particular task merely to examine how he or she handles it.

2.3 Computer Security Issues

The anomaly detection domain has been an area of active study within the computer and information security communities since Anderson's introduction of the concept in 1980 [5], and Denning's formal model in 1987 [17]. Much of the research has been explicitly oriented toward the assumption of an intruder penetrating the computer system from outside, but more modern investigations have broader scopes. Kumar, for example, defines anomaly detection as follows: "Anomaly detection attempts to quantify the usual or acceptable behavior and flags other irregular behavior as potentially intrusive" [28]. Under this definition, the scope of anomaly detection encompasses not only violations by an outsider but also anomalies arising from violations on the part of an authorized user (the *trusted insider* threat). Current anomaly detection models consist primarily of expert system rule bases and statistical profiles, though the emphasis on the various components differs from system to system.

2.3.1 Rule-Based Detectors

One approach to misuse detection is to develop a rule base which describes all possible misuse scenarios and to then employ some form of pattern matcher (such as an expert system) to activate appropriate rules. This form of detection is appealing because it can detect patterns of attack quickly as it doesn't depend on aggregate data that must be acquired over an extended time period such as statistics or behavioral profiles. In addition, known attacks can be reliably detected and differentiated from valid usage without the uncertainty associated with statistical measures.

Unfortunately, the very idea of misuse or abuse is fundamentally tied to that of *security policy* [27], and policy is decided by humans — often in a distinctly non-formal manner. Furthermore, as the number of possible states that a computer system can be in is truly vast (and changes with each reconfiguration of the system hardware), completely partitioning this space into 'abusive' and 'normal' can be a difficult if not impossible task. Thus, specifying a complete and accurate definition of 'misuse' or 'acceptable use' is impractical. Current systems, therefore, rely on matching signatures of known attacks, generated by hand from the experience of human operators. This is both labor intensive and suffers from inability to detect previously unknown attack patterns. Nonetheless, some extant systems rely primarily on such rule bases for anomaly detection. Purdue's IDIOT system [29, 28] uses colored petri net models as attack patterns, while the GrIDS [65, 14] system employs subgraph matching rules to examine network interconnection graphs. Such models are quite expressive and powerful, but, to date, the attack pattern models must still be generated by hand. Signature based detectors *have* proved to be quite useful in diagnosing known security vulnerabilities (for example, the SATAN/SANTA tool [18]) or virus attacks (for example, [22]).

More often rule bases form a single component of an anomaly detection system. Systems such as (N)IDES [35, 34, 26, 3] and its successor EMERALD [43], AIS [25], MIDAS [58], and NSM [24] all employ rule bases higher as level decision procedures in a hierarchical detection system. These procedures are used as discriminators to prune out spurious hits from lower level sensors and anomaly detectors which may be statistical in nature. Even for such systems, though, it is desirable to increase the accuracy of hits passed up the detection hierarchy from the leaf level detectors.

2.3.2 Statistical Detectors

Denning formalized the concept of adaptive, statistical anomaly detection, [17]. She presented the anomaly detection framework that was developed into SRI's (N)IDES detector, and which has since been adopted for use in a number of different detection systems. Denning proposed that the user profile be formed of a statistical model of behavior measured across a variety of metrics extracted from audit records such as number of password failures per minute, time between logins, or pages printed per day. She presented the following five potential statistical models for summarizing the metrics into a profile:

Operational Model This model employs fixed, empirically selected, thresholds for a metric. An observation is abnormal if it falls outside these thresholds.

Mean and Standard Deviation An observation is abnormal if it falls outside a confidence interval about the mean.

Multivariate This is the multivariate extension of the mean/standard deviation model, employing an n dimensional mean vector and the corresponding covariance matrix.

Markov Process[1] This models each type of audit record as a state variable and defines transition probabilities between them. An observation is abnormal if its probability of occurrence after the observed predecessor is too low.

Time Series This model employs the inter-arrival times of events to estimate the probability of occurrence of a newly arrived event. In the general case, this model includes the complete set of marginal densities, $P(O_t|O_{t-1}, O_{t-2}, \ldots, O_0)$ and is of exponential complexity. It is, therefore, necessary to impose some constraints on this model for the sake of practicality.

[1]It is important to note that the Markov process model presented by Denning is distinct from the hidden Markov model we describe in Section 2.2. Denning's Markov process model assumes that the state variables correspond to some real and observable phenomenon (audit records, in this case), while an HMM assumes that the state variables are hidden and correspond to phenomena that are, perhaps, fundamentally unobservable (such as a user's mental states).

Many current anomaly detection systems employ only the first three of Denning's models. HAYSTACK [59, 39], for example, models a number of system parameters as independent, Gaussianly distributed random variables. Individual parameters are considered to be abnormal when they fall outside the 90% data range for that variable. Total profile abnormality is calculated as a weighted sum over the abnormal parameters. NSM [24] models network activity as a four dimensional sparse matrix where each point represents a particular network connection. Abnormality is detected either with an expert system rule, or by comparison against an 'expected behavior' matrix mask. The expected behavior matrix is generated from accumulated statistics of the complete local network over time.

To our knowledge, the Markov process statistical model has remained theoretical since Denning's proposal, as has the full (exponentially complex) time series analysis method. Some recent anomaly detection work has focused on time series analysis by considering various restrictions to the general method. We examine these techniques in the next section.

2.3.3 Time Sequence Anomaly Detection Models

In recent years, work has appeared which extends the anomaly detection model away from the rule bases and first order statistics that have characterized many well known anomaly and intrusion detection systems. While these approaches can, in principle, be assimilated into the very broad model of anomaly detection presented by Denning, they represent a dramatic departure from the majority of techniques previously developed.

Forrest et al. [20, 19] have examined the problem of detecting anomalous actions on the parts of privileged system programs (daemons, in the UNIX operating system parlance). Such programs have been widely exploited by malicious users and intruders to elevate privilege or tamper with data. Forrest's approach tracks the execution traces of these programs at the system call level, comparing fixed length temporal sequences of events to a dictionary of expected behaviors for that program. Because monitoring of system calls must take place extremely quickly and with little overhead, they employ an exact string match for a comparison function. Exact matching is a plausible technique for examining sequences of system calls, which are typically generated by a compiler, but is generally insufficient to examining sequences of user generated events. We have found that humans rarely tend to *exactly* repeat prior sequences of actions.

An alternative method of employing sequence learning to the anomaly detection domain was presented by Teng et al. in [67, 68]. Their system develops sequential rules of the form '**E1 - E2 - * - E3 $--$ > (E4 = 94%; E5 = 6%)**', where the various **E**'s are events derived from the security audit trail, * denotes 'any event', and the percentages on the right hand of the rule represent the probability of occurrence of each of the consequent events given the occurrence of the antecedent sequence. The rules are generated inductively with an information theoretic algorithm that measures the applicability of rules in terms of coverage and predictive power of each rule. While the temporal aspects of this work are similar in nature to our own, Teng et al. have not, to our knowledge, examined the continual learning and concept drift aspects associated with adaptive anomaly detection.

2.3.4 The Insider Threat

Although many security techniques are oriented toward defending the system perimeter, thus preventing intrusion by an outsider, there is a strong need for the converse: the ability to defend against malicious actions on the part of already authorized users. The 1996 Ernst & Young Information Security Survey [73] reports that, of companies who suffered financial losses from problems with information security and disaster recovery, "A third (42% at larger companies [those with more than 1,000 employees]) cited malicious acts by company insiders," as the security violations. The statistic may be even higher; according to Boedges [8] (as reported in [59]), 80% of computer crimes known to the Air Force Office of Special Investigations Computer Crimes Division were carried out by trusted employees.

While the anomaly detection concept was originally formulated as a defense against intruders, the framework can also, in principle, support monitoring of authorized users for unusual behavioral patterns. When hostile or abusive actions require drastic or highly unusual activities, they may be subject to detection by an anomaly detection agent. For example, massive file deletes or copies or execution of security penetration codes on the part of a 'normal' user may well be part of a hostile action. The large difficulty is in differentiating such hostilities from innocuous and expected change on the part of the user. Does the use of a command previously not seen for the profiled user signal the start of an attack sequence, or merely that the user read the system manual and has become more proficient? Adaptability on the part of the anomaly detection system is desirable for the latter purpose, but may defeat the utility of the system for detecting the former. This difficulty has traditionally been approached through the addition of attack signature rule bases to the adaptive component to form a hybrid anomaly detection system, as described previously.

2.3.5 Hostile Training

Another threat facing adaptive anomaly detection systems is that of *hostile training*. An informed intruder or masquerader (one aware of the anomaly detection system and the content of one or more user profiles) may be able to emulate the behaviors of a valid user well enough to pass unnoticed. Over time, the hostile user can alter behavior, presenting a gradual change to which the anomaly detection system may adapt. HAYSTACK approaches this problem by profiling each user in two ways: as an individual and as part of a class of users [59]. The hypothesis is that while a single user may be able to train the individual profile to accept abusive behavior, but will not be able to distort the entire class's profile (assuming the class is comprised of sufficiently many users). In practice, HAYSTACK's class profile is fixed and represents "generic notions of acceptable behavior for a group of users."

2.3.6 System Initialization

A difficulty of adaptive approaches to anomaly detection is defining a policy for handling startup situations. The system must have some knowledge of the user or system in question before it can make reliable classifications, but initially (when encountering a user for the first time) no such knowledge is available. Furthermore, initial acquisition of user data is as subject to subversion as later data, but the anomaly detector is not yet in a position to detect attempted subversion.

HAYSTACK [59, 39] handles startup conditions by initially treating users as members of a behavioral class. The class profile is constructed by hand from observation of the class in question and has relatively broad definitions of acceptable behavior. So long as the new user adheres to the class profile, the user is taken to be valid. The user's profile is initialized to a copy of the class profile and, over time, is specialized to fit that user.

Although the startup problem is a critical one, with important implications for the practical use of adaptive anomaly detection mechanisms, we do not intend to address it within the scope of the proposed research.

Chapter 3

Current Results

In this chapter, we describe our current prototype anomaly detection system and present an empirical analysis of its behavior. We close with our current theoretical results on the structure of concept drift.

3.1 Description of the Current Experimental System

Following [16], we perform anomaly detection by reference to a known *user profile*, built from historical command token data from the user of interest. Once a user profile is formed, the basic action of the detection system is to compare incoming input sequences to the historical data and form an opinion as to whether or not they both represent the same user. The fundamental unit of comparison in the anomaly detector system is the command sequence. To this end, all input token streams are segmented into overlapping sequences of tokens (where the *length* of each sequence is a parameter to the system, but is fixed for a single run). Sequences are compared to one another using one of a variety of possible similarity measures. Historical sequences, known to belong to the valid user, are stored in a sequence *dictionary* which, along with various other parameters such as sequence length, comprise a user's profile.

An overview of the data flow in our prototype anomaly detection system is shown in Figure 3.1. The stream of tokenized command line data (left) is compared to the user's profile via a similarity measure function Sim() to yield a raw similarity stream. A noise suppression function, $F()$, is employed to

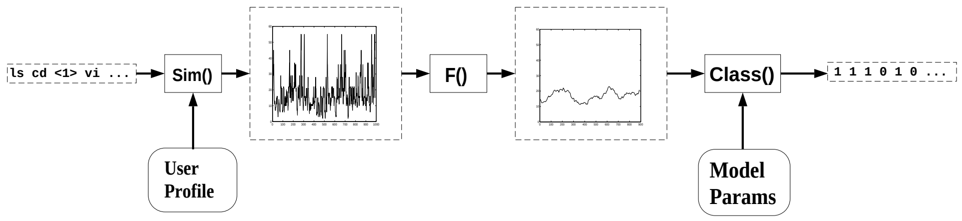


Figure 3.1: Prototype anomaly detection system data flow overview

smooth the raw similarity stream and improve classification accuracy. Finally, the smoothed similarity stream is classified with respect to model parameters (which may be fixed or learned) to produce a classification stream. The classification stream indicates, at each time step, whether the current user's behavior is normal or anomalous.

3.1.1 Data Collection and Parsing

To learn characteristic patterns of actions, our system uses the *sequence* (an ordered, fixed-length set of temporally adjacent actions) as the fundamental unit of comparison. For this research, actions were taken to be UNIX shell commands with their arguments, although the approach developed here is general and can be extended to any stream of discrete events or event vectors such as operating system calls or graphical user interface events. For ease of data collection, the temporal order of commands was maintained only within the context of a single command interpreter (a shell). Currently, we preserve command names and argument switches but omit the specific file names associated with each command execution. This decision was based on the intuition that the significant facet of the user's command history for this work was *behavior* rather than *content*. Thus, it should be more useful to note that the user invoked the command `emacs` (a text editor) with the behavioral switch `-nw` (run in text-mode rather than initialize the X-windows interface) and two file names, than it would be to take note of the actual file names used. Clearly, for some applications of misuse detection, important information could be extracted from the filenames (directories in which the user typically works, for example).

To collect user action data, we created a parser for the UNIX `csh` family of languages (including `tcsh`) which translates the raw data stream of the shell command trace into a token stream suitable for storage and comparison. This translation suppresses filenames, as described above, but preserves command names, argument switches[1], and other syntactically important symbols such as `|`, `;`, and `>&!`. For example, the command stream:

```
> ls -laF
> cd /tmp
> gunzip -c foo.tar.gz | (cd \~ ; tar xf -)
```

would be translated by the parser into the token stream:

```
ls -laF cd <1> gunzip -c <1> | ( cd <1> ; tar - <1> )
```

where the token `<1>` denotes the occurrence of a single filename argument[2]. The parser also introduces the tokens `**SOF**` and `**EOF**` indicating start and end of a command interpreter session, respectively.

3.1.2 Sequence Comparison via Similarity Measures

A number of possible methods exist for measuring the similarity of two sequences. The most straightforward is the equality function, which yields `TRUE` when both sequences match in every position and `FALSE` otherwise. This is the similarity function employed by string matching algorithms and has the advantage of being widely studied and highly optimizable. For example, the UNIX `diff` program employs this form of matching. When examining user command traces, the difficulty arises that for long sequences the probability of locating exact matches in historical command data becomes exceedingly low (in one series of tests, only 7% of a user's command sequences exactly matched some historical command sequence). Thus, the equality function is not a viable choice for this particular domain.

Our system, therefore, computes a numerical *similarity measure* that returns a high value for pairs of sequences that it believes to have close resemblance (bounded above by a constant, $\mathrm{Sim}_{\max}$, defined as the similarity

[1]Strictly speaking, the parser depends upon the UNIX convention that argument switches are prefixed with a dash, so the '`-laF`' switch in the command `ls -laF ${HOME}` would be correctly recognized, but the switch `tvf` in the command `tar tvf /tmp/foo.tar` would not be. This is not taken to be a serious weakness, however, as the dash convention is widely used.

[2]Multiple filenames are replaced by an appropriately numbered token. For example, the parser would emit a set of five filename arguments as `<5>`

of identical sequences), and a low value to pairs of sequences that it believes largely differ (bounded below by 0 for sequences having no elements in common). The individual elements of the sequences are from an unordered set, which creates a matching problem similar to that of symbolic features for IBL. However, unlike IBL, our similarity measure is judging the similarity between two *sequences* rather than two feature vectors. The temporal nature of sequence matching suggests that interrelations between different elements of the sequence vector can be important.[3]

Initially, we examined a similarity measure that simply assigns a score equal to the number of identical tokens found in the same locations of the two sequences. Upon consideration, however, we hypothesized that a measure that assigns a higher score to adjacent identical tokens than to separated identical tokens might be preferable. The intuition is that token matches separated by interleaving non-matching tokens are more likely to have occurred by chance, while adjacent matches are more likely to have occurred due to a causal process. Therefore, if sequence Seq_1 has k tokens in common with each of Seq_2 and Seq_3, but the common tokens are adjacent in Seq_1 and Seq_2 then we would like the similarity measure to have the property that $\mathrm{Sim}(Seq_1, Seq_2) > \mathrm{Sim}(Seq_1, Seq_3)$. Under this requirement, the pair of sequences shown below on the left would have a higher similarity value than would the pair on the right.

```
ls foo ; vi                      ls -l foo ;
ls foo cat bar                   ls -F foo cat
```

Thus, one axis of differentiation between similarity measures is 'does not detect match adjacency' versus 'detects match adjacency'. A second axis is the bound of the maximum similarity measure as a function of sequence length. The similarity measure that scores sequences from a count of matches, regardless of adjacency, has an upper bound that is polynomial in the length of the sequences. Specifically, for sequences of length n, this measure is $\leq n$. We denote this similarity measure as MC-P (for match count with polynomial bound). To examine the hypothesis that detecting match adjacency is useful for this task, we modified MC-P to bias the similarity score in favor

[3]Note that the possibility of interdependence between vector elements implies that the similarity function is unlikely to be metric, that is, we cannot assume that the triangle inequalitity holds.

of adjacent matches (as described below). This measure is denoted MCA-P (for match count with adjacency and polynomial bound) and is bounded by $n(n+1)/2$. A polynomial bound seems appropriate, considering that the central hypothesis is that adjacent tokens are produced by a (mostly) causal process, and, therefore, should display a high degree of correlation.

In a stream of independently generated tokens, it seems likely that an exponentially bounded function would be more appropriate. Our intuition is that the causal linkage of user-generated tokens will evidence itself as a deviation from the characteristics of the independence assumption. To test our hypothesis (i.e. reject the independence hypothesis), we examined the behavior of exponentially bounded measures analogous to the polynomially bounded ones just described. The MC-P measure was modified to score exponentially in the number of matches (still without considering adjacency) and was labeled MC-E. The MC-E measure has upper bound 2^n. The MCA-P measure was adapted in a similar fashion to create MCA-E, whose upper bound is $2^n - 1$. All four similarity measures are encompassed by the algorithm of Figure 3.2 operating on sequences Seq_1 and Seq_2.

```
1. Set an adjacency counter, $c := 1$ and the value of the measure, Sim := 0.
2. For each position, $i$, in the sequence length, $n$:
    • If $Seq_1(i) = Seq_2(i)$ then Sim := $f(\text{Sim}, c)$ and $c := u(c)$
    • Otherwise, $c := 1$.
3. After all positions are examined, return the measure value.
```

Figure 3.2: Sequence similarity calculation algorithm

The differences between the measures are determined by the nature of the scoring function, $f(\text{Sim}, c)$, and the adjacency update function, $u(c)$, as summarized in Table 3.1.

Finally, we define the similarity of a single sequence Seq_i to a set of sequences, D, as:

$$\text{Sim}(Seq_i, D) = \max_{Seq_j \in D} \{\text{Sim}(Seq_i, Seq_j)\}$$

	$f(\text{Sim}, c)$	$u(c)$
MC-P	$\text{Sim} + 1$	1
MCA-P	$\text{Sim} + c$	$c + 1$
MC-E	$2 \cdot \text{Sim}$	1
MCA-E	$\text{Sim} + c$	$2 \cdot c$

Table 3.1: Scoring and update functions for the similarity measures

Thus, the similarity of a sequence to the user dictionary is the measure of that sequence compared to the *most* similar sequence in the dictionary. This is related to the IBL 1-nearest-neighbor classification rule, according to which an instance is classified only by the closest instance in the learned model.

3.1.3 Instance Selection and Dictionary Pruning

The potential amount of data that is available for examination on a per-user basis is staggering. If the granularity of examination is reduced to the level of individual operating system calls, the data stream could well amount to thousands, or even millions, of data items per second. If the anomaly-detection system is to run real-time then it is imperative that the system be both fast and resource conservative (history has shown that a security measure that is sufficiently obtrusive will not actually be used, and will, therefore, be useless—for example, [66]). Thus, much of the available data must be discarded with little or no examination. To this end, after initial dictionary construction, we reduce the dictionary to a fixed size through *pruning* according to a utility heuristic. The final dictionary size is a parameter of the user profile, and is currently set empirically.

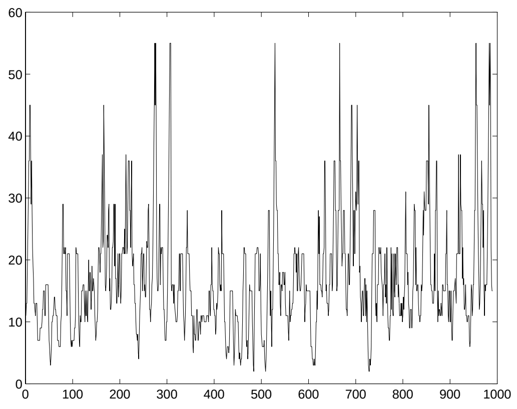
We have investigated a number of pruning heuristics for this domain. Under the definition of similarity to a set, we note that only a single dictionary sequence can be selected as most similar to an input sequence under investigation. If we assume that the characteristics of a user's behavior change relatively slowly, we can invoke locality of reference to predict that recently matched dictionary sequences will be used again for detection in the near future. This suggests an analogy to tasks in operating systems, such as page replacement, in which some resources must be discarded in favor of others. This analogy led us to the LRU (least recently used) pruning heuristic. Under

Name	Removal Criterion	Heuristic
LRU	least recently used	t_a
LFU	least frequently used	c_a
WLFU	weighted LFU	$c_a 2^{-t_a}$
truncate	newest	$\frac{1}{t_d}$
FIFO	oldest	t_d
random	random	`random()`

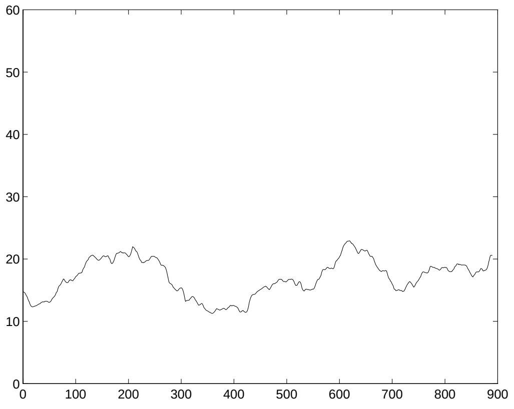
Table 3.2: Summary of dictionary pruning heuristics

this strategy, the parameter selection data set (separate from both train and test data sets) are used to mark which sequences in the dictionary are used for detection. As each pruning sequence is examined, the dictionary instance selected as most similar is time-stamped. After all pruning data is processed, the dictionary is reduced to the desired size by removing the least-recently-used sequences (where sequences that are never matched by the similarity measure are taken to be older than all other sequences). The resulting pruned dictionary is then employed as the user profile to classify input streams. Strictly speaking, LRU is an iterative algorithm, while our implementation for these experiments was batch mode. Note, also, that the amount of parameterization data can impact the performance of the LRU algorithm. If the parameterization data contains fewer instances than does the training data, for example, there *must* exist some instances that have not been marked as contributing to classification. LRU can make no decisions about such sequences, as they have no timestamp.

Finding the LRU heuristic to be useful, we proceeded to implement and test the LFU (least frequently used: remove sequences with smallest use count) and weighted LFU (use count weighted by an exponentially decayed age factor) heuristics. For comparison, we also included truncate (remove most recent sequences), FIFO (remove oldest sequences), and random (remove sequences randomly) pruning policies. All the currently implemented pruning heuristics are summarized in Table 3.2. Heuristics in this table are described in terms of the parameters t_d (time sequence was stored into the dictionary), t_a (last access time—i.e. the last time at which this sequence was selected as most similar to an input sequence), and c_a (access count).



(a)



(b)

Figure 3.3: Similarity measure stream. **(a)** Raw. **(b)** Smoothed.

3.1.4 User Verification

Given an input stream of command tokens parsed by the data collection module, the detection module classifies the current user as normal or anomalous after each token. The output of the detection module is a stream of binary decisions indicating, at each point in the input command data, whether or not it believes that the input stream at that point was generated by the profiled user.

To make these decisions, the detection module first calculates the similarity of each input sequence to the user's dictionary, yielding a stream of similarity measures. In an intuitive sense, this stream represents the familiarity of the input commands at each time step, given knowledge about the previous behavior of the user. In preliminary experiments, we discovered that the similarity value stream produced by comparison of test data to a user profile was noisy and erratic (see Figure 3.3, **(a)**). The noisiness of the raw similarity measure stream can be attributed to normal deviations in actions on the parts of the users, as well as to random elements (preempting work to deal with urgent e-mail, for example). Although explainable, this variance in the similarity measure makes it impossible to detect anomalous behavior from a single sequence. (The profiled user sporadically has very low similarity with their own past behavior.)

Based on the hypothesis that, while individual sequences may deviate from historical precedent, aggregate behavior should largely conform to historical behavior for valid users but should still noticeably deviate for intruders, we applied a smoothing filter to the data (see Figure 3.3, **(b)**). The smoothing filter we applied was a windowed mean-value filter, which at sequence i of the input stream is defined by:

$$m_w(i, D) = \frac{1}{w} \sum_{j=i-w}^{i} \mathrm{Sim}(Seq_j, D)$$

where D is the user profile dictionary and w is the window length. This filter suppresses high frequency noise effects at the cost of detection speed. It is thus desirable to choose the minimum window length capable of sufficient noise suppression. Currently, the window length is selected empirically from observations of behavior over a subset of our user population. The curves in Figure 3.4 display the effects of window lengths for USER3's profile. Each point on each curve represents the average similarity to USER3's profile for all sequences up to and including that time. Thus, the curves display the net effects of successively longer windows. For sufficiently long windows, the average similarity converges to the true mean with a small variance. At these long window lengths, a large separation between USER3 (the top curve) and other users is observed. Unfortunately, such long windows entail large time lags before detection, so we chose the smallest window that produced visible separation (80 sequences, for these experiments).

After smoothing the similarity measure stream, the detection module makes a classification of the input stream as being normal or abnormal at the point occurring at the end of the current window. A Bayes-optimal classifier is not available to us, because we have training data only for the profiled data. As a result, we can characterize the distribution of similarity measures between a user and his or her profile, but not the similarity distribution between that profile and an intruder. Furthermore, we have found that a hypothetical Bayes-optimal classifier (not weighted for classification costs), is overly critical of the profiled user. Figure 3.5 displays an *a posteriori*Bayes-optimal classifier for USER6's test data when measured against USER3's profile. In this case, the (unweighted) Bayes-optimal boundary yields far too high a false error rate.

In light of these considerations, the classification mechanisms we employ are based on threshold tests. The first method we examined used a single threshold for all users selected by observation on a subset of the user population. This test can be viewed as a case of the operational statistical model presented

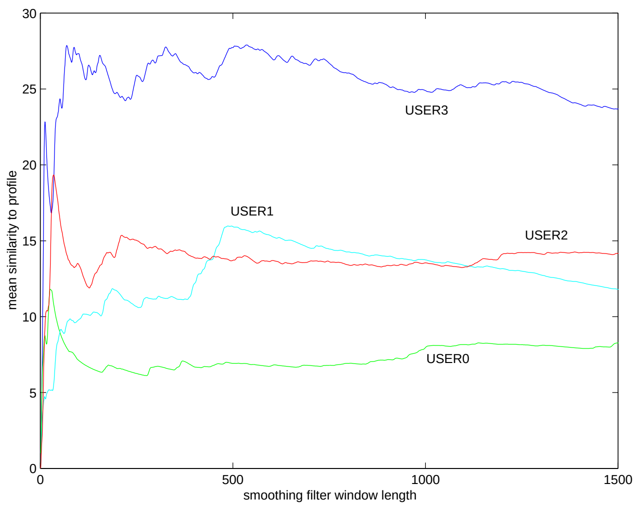


Figure 3.4: Window length effects for USER3's profile

by Denning in [17]. When the average similarity to the profile fell below the threshold the sample was considered to be anomalous, otherwise it was considered to be normal. We have also examined using dual thresholds (a floor and ceiling) such that values falling between t_{floor} and t_{ceil} are considered to be normal and samples outside this range are considered to be anomalous. The thresholds were selected in two ways: for early experiments, they were set by hand to a single value for all users, by examining long-term average similarities. For example, for the data displayed in Figure 3.4, a lower threshold of 15 and an upper threshold of 30 are appropriate thresholds. For later experiments, the thresholds were automatically selected on a per-profile basis by examination of an independent *parameter selection set* of command sequences. Thresholds were chosen to achieve a specified classification accuracy on the parameter data.

We have also briefly investigated the possibility of classification through clustering along the similarity axis with Parzen windows [21]. For the similarity measures we are currently employing, we found that the data tends to develop only a single window and that this window can be well approximated by the floor and ceiling threshold system previously described.

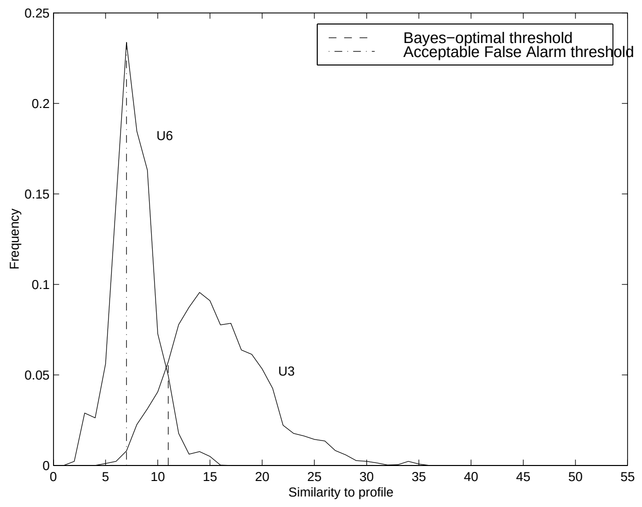


Figure 3.5: Comparison of Bayes-optimal decision boundary and acceptable false alarm rate boundary. The rightmost curve (user U3) represents the profiled user.

3.2 Experimental Results

In this section, we present empirical evaluations of our prototype anomaly detection system. We describe the sources and structure of the data employed, and give experimental results showing the performance of the system and the impact of various parameter settings.

3.2.1 Experimental Structure

The data used in these preliminary investigations were command histories collected from five members of the Purdue MILLENNIUM lab over the course of slightly more than an academic semester and two sets donated by other students. Command histories were generated from the `tcsh` UNIX shell running under Solaris 2.5 or Linux 2.0. Command data were gathered with the full knowledge and consent of the participating students, which may have affected their behavior during the period of data collection. Possibly a more serious impact on the structure of the data, however, is the fact that all donating students were fairly to highly experienced computer users. We are currently attempting to obtain command data from a variety of sources, including novice

User	# Tokens
USER0	8,001
USER1	25,545
USER2	13,164
USER3	25,891
USER4	4,104
USER5	1,593
USER6	4,400

Table 3.3: Tokens available for experimentation per user

computer users, to help assess the generalization abilities of the techniques reported here to other segments of the user population.

The number of tokens available for experimentation differed on a per-user basis (because of different work patterns, tasks, typing rates, data collection period, etc.). Users 4, 5, and 6, for example, began contributing data substantially later than the other users and are less well represented. For the experiments reported in this section, the number of tokens available for each user is reported in Table 3.3.

Because of the (relatively) low token count from users 4, 5, and 6, they were generally only used for testing purposes while profiling was done on users 0–3. The one thousand most recent tokens were reserved from each data set for testing purposes. For the profiled users, the remaining data was split into train and parameter selection sets at a proportion of 2/3 to 1/3, with the train data selected from the oldest tokens. Thus, for the experiments performed here, the train and test data are separated by a variable amount of time and the results reported here may well reflect concept drift. Because the experiments were performed concurrently with data acquisition, and different users contributed data at different times, not all users were available for all experiments.

The assumption was made for the experiments reported in this paper that all training data was pure—that is, that each user's data reflects only that user's actions and not the actions of an intruder or another user masquerading as that user. Although this assumption is difficult to verify, we currently have no reason to doubt it.

3.2.2 Testing Procedure

Because we lack data of actual intrusions or abuses, we test our system on synthesized anomalous situations. For each available user (for whom we have sufficient data) we create a user profile as described in Section 3.1, employing a fraction of their data. The remaining data from each user is then used as test data, and is tested against all profiles. When the test data and profile originate with the same user, the test measures the system's ability to recognize the valid user (or, conversely, the false alarm error rate). When the test data and profile originate with *different* users, the test measures the system's ability to detect an anomalous or intrusive situation (and the corresponding false acceptance error rate). This form of synthetic attack represents only one particular kind of anomalous situation — that of a naive intruder gaining unauthorized access to an account — but it allows us to characterize the baseline performance of our system.

3.2.3 Proof of Concept: User Differentiation via Sequences is Possible

A central hypothesis of our anomaly detection system is that user patterns are sufficiently consistent, for a single user, yet sufficiently disparate, when measured between users, that differentiation is possible. Furthermore, we propose that this differentiation can be made through the use of temporal sequences of commands. In this experiment we show that differentiation is, in fact, possible within the scope of our test data. Table 3.4 displays the detection results for all pairwise tests of user test sets and available user profiles with sequence length of 12 and a dictionary size of 2000 sequences using the MCA-P similarity measure (as described in Section 3.1.2).

The user from whom the profile was generated is listed in the leftmost column, while the user from whom the test data was generated is listed across the topmost row. The numbers in the table are percentages of windows that the detection system identified as the profiled user. Ideally, the diagonal elements of the table (true positive rates) should be 100% and the off-diagonal elements (false positive rates) should be 0%.

These results demonstrate that the recognition system has higher true positive than false positive rates. Furthermore, in some cases (USER1 tested against USER0 and USER2 tested against USER1, for example), the false

Profiled User	Tested User			
	USER0	USER1	USER2	USER3
USER0	99.19	35.35	6.113	0.000
USER1	17.84	88.30	23.32	1.251
USER2	3.519	54.86	72.10	8.292
USER3	6.270	15.74	11.52	69.85

Table 3.4: Proof of Concept: All users vs. all profiles.

negative rate is lower than the false positive rate. This is a desirable characteristic as false negatives make the system less usable (due to the annoyance of false alarms). We take these results as evidence that the closed world assumption is appropriate for at least some users in this domain, although we note that the users involved in this study are all fairly to extremely experienced computer users. The question of whether or not the techniques presented here would apply equally well to novice users is still open.

3.2.4 Exploration of the Effects of Dictionary Size

Early investigations led us to examine the question of whether optimal dictionary size is invariant of the user profiled. Tables 3.5 **(a)**-**(d)** suggest that ideal dictionary size is user specific. In these tables, the column headings indicate the size of the dictionary used in the user profile (selected from all available training data with the truncate pruning heuristic, as defined in Section 3.1.3), while the row headings indicate the test set under examination. 'SELF' denotes a test of the user's data against that same user's profile. The numbers in the table are percentages of the input stream detected as the same as the profile. Thus, the ideal values for the 'SELF' row are 100% and the ideal values for other rows are 0%.

It appears that, while for USER0 most of the important behavioral data is extracted within 500 sequences, for USER3 significant information is still being acquired by the 1000 and 2000 sequence points. USER2 and USER3's accuracies do not appear to asymptote on this range of sequences. There are a two possible explanations for this behavior. The first is that users 1, 2, and 3 are also characterized by a small number of sequences, but that those sequences occur infrequently, and thus require a larger sample to acquire. Under this hypothesis, it is possible that a single dictionary size is applicable to all users,

	200	500	1000	2000
SELF	19.1	92.1	98.8	99.2
USER1	9.0	12.90	27.6	35.4
USER2	0.0	0.0	2.0	6.1
USER3	0.0	0.0	0.0	0.0

(a)

	200	500	1000	2000
SELF	46.0	74.4	82.6	88.3
USER0	0.0	7.4	12.0	17.8
USER2	4.7	10.1	15.6	23.3
USER3	0.0	1.0	1.0	1.3

(b)

	200	500	1000	2000
SELF	4.6	21.7	55.0	72.1
USER0	0.0	1.5	2.8	3.5
USER1	8.3	22.4	46.1	54.9
USER3	0.0	0.2	1.4	8.3

(c)

	200	500	1000	2000
SELF	14.3	36.9	55.3	69.8
USER0	1.4	1.8	2.9	6.3
USER1	0.5	2.4	6.1	15.7
USER2	0.0	2.1	5.6	11.5

(d)

Table 3.5: Profiled users (SELF) versus all other users for various dictionary sizes.

and that the important sequences occurred early for USER0 more-or-less by chance. Alternatively, it is possible that different dictionary sizes are necessary for superior performance for different users. This issue is complicated by the result that the false positive rate seems to increase with increasing dictionary size; it is desirable to maintain the smallest acceptable dictionary for accuracy as well as resource reasons.

3.2.5 Investigations Into the Problem of Instance Selection

We investigated the behavior of the dictionary pruning heuristics defined in Section 3.1.3. Characteristic results for the LRU heuristic are given in Tables 3.6 **(a)**-**(d)**. The format of these tables is identical to the format of Tables 3.5 **(a)**-**(d)**.

For USER2 and USER3, these results display a dramatic increase in true detection rate, accompanied by a decrease in false positive rate. USER0, on the other hand, experiences a slight drop in true positive rate for large dictionary sizes, along with increases in false negative rate in some cases. There are two noteworthy features of USER0's results. The first is that the true detection rate (the SELF rate) for a dictionary size of 200 elements is much greater than

	200	500	1000	2000
SELF	69.4	86.8	90.7	93.4
USER1	13.3	37.6	46.7	50.9
USER2	0.0	0.0	0.0	0.0
USER3	0.0	0.0	0.0	0.0

(a)

	200	500	1000	2000
SELF	28.3	89.8	100.0	100.0
USER0	2.1	15.4	25.8	36.7
USER2	0.0	1.3	8.1	11.4
USER3	0.0	0.0	0.0	0.0

(b)

	200	500	1000	2000
SELF	10.7	47.1	70.1	95.0
USER0	2.2	3.1	3.4	10.6
USER1	1.8	16.0	42.8	60.1
USER3	0.0	0.0	0.0	0.9

(c)

	200	500	1000	2000
SELF	32.3	69.9	80.4	89.8
USER0	0.0	1.2	2.9	5.9
USER1	0.0	0.0	0.0	0.2
USER2	0.0	0.0	3.2	3.6

(d)

Table 3.6: Results of LRU instance selection

for the equivalent entry in Table 3.5 (improved by 50 percentage points). This suggests that the LRU pruning algorithm can be useful even for this user. It also suggests, however, that the LRU selection technique might asymptote at lower accuracies for the other users as well, if the experiments were extended to cover larger dictionary sizes. The second interesting factor in Table 3.6 for USER0 is that, while false positive rates increased with respect to USER1, they declined or remained constant with respect to USER2 and USER3. This raises the possibility that the optimal instance selection scheme is not merely a function of the user being profiled, but also of the intruder.

3.2.6 On the Choice of Similarity Measures

As described in Section 3.1.2, an important parameter of our anomaly detection system is the similarity measure employed for sequence matching. Currently, we have examined the behaviors of four possible sequence similarity measures, differentiated by growth rate (polynomial or exponential in number of tokens matched) and match locality (whether or not match adjacency is factored into the final similarity measure). To examine the degree of class separation produced by each of these measures, we used each to classify user command history traces. From each user's data set, two thirds of the tokens were devoted to training (i.e. initial dictionary construction) and the remaining one third was divided into 1000 testing instances and the rest into instances used for dictionary instance selection and parameter selection. The user profiles were initialized with all available training data and were pruned down to the desired testing sizes of 200, 500, or 1000 sequences via the LRU instance selection algorithm as described in Section 3.1.3.

All experiments employed a sequence length of ten tokens and a smoothing window length of eighty sequences. Profiles were created for four of the users and then the test data from all seven users were classified against the profiles according to each similarity measure. The classification thresholds were selected to achieve a false-negative error rate of 1% on the parameter selection data (as detailed in Section 3.1.4). Results of the experiments are displayed in Tables 3.7 and 3.8. The results given here are typical and are generally reflective of the trends we found in the data.

	MC-P	MCA-P	MC-E	MCA-E
USER0	93.96	67.07	73.44	79.36
USER1	9.11	0.00	0.00	8.23
USER2	0.00	0.00	0.00	0.00
USER3	15.37	0.00	0.00	0.00
USER4	5.49	0.00	0.00	0.00
USER5	10.54	0.00	0.00	0.00
USER6	10.87	0.00	0.00	0.00

Table 3.7: Detections over all users and similarity measures for USER0's profile, dictionary of 1000 sequences

	MC-P	MCA-P	MC-E	MCA-E
USER0	60.92	33.48	52.69	29.75 †
USER1	68.72	70.14 †	77.06	73.77
USER2	93.85	99.89	99.89	99.89
USER3	61.25	48.30 †	52.47	54.99
USER4	34.80	33.26 †	39.08	35.13
USER5	79.69	75.74 †	87.16	87.38
USER6	81.78	70.58	83.53	68.06 †

Table 3.8: Detections over all users and similarity measures for USER2's profile, dictionary of 200 sequences

Each value in these tables reports the percentage of instances for which the tested user was identified as the profiled user. The goal, therefore, is to minimize all rows other than the profiled user and to maximize the row corresponding to the profiled user (USER0, in Table 3.7 and USER2, in Table 3.8). In Table 3.8, the symbol '†' indicates the position in which the best value occurs for each user. As above, MC and MCA denote match count and match count with adjacency bias similarity measures, respectively, while the suffixes P and E indicate that the similarity measure's upper bound is polynomial or exponential in the length of the sequence.

Table 3.7 indicates a definite superiority of MC-P for the task of identifying the profiled user (USER0) but somewhat inferior behavior on the task of distinguishing other users from USER0. By contrast, all other algorithms perform spectacularly when distinguishing other users from the profiled user but have what is likely to be an unacceptably high false negative rate. Unfortunately, the behaviors of many of the similarity measures are nearly identical for large parts of the tested space, so it is difficult to identify their relative merits from this data. Nonetheless, for USER0's profile, the MC-P similarity measure seems to be preferable. This result provides evidence against the hypothesis that detecting adjacent matches is desirable.

The results in Table 3.8 indicate that, overall, MCA-P is the preferable similarity measure for USER2's profile, followed by MCA-E. A strong preference for the adjacency-based measures is supportive of the hypothesis of the causality of command sequences. And, as with USER0, preference for the polynomial bounded similarity measure further supports this hypothesis. It's also noteworthy that the MC-P algorithm has the poorest performance for this user (often dramatically so). This indicates that there are cases in which equality matching (with the addition of 'don't care' positions) is insufficient to the task of user modeling in this context.

Together the results for USER0 and USER2 demonstrate that different similarity measures are appropriate for different users. This indicates that there is a need for a method to detect which similarity measure is appropriate for a particular user and that future research should take this into account. This issue is further complicated by the possibility that the optimal similarity measure may be time variant within the context of a single user. Finally, the possibility exists (and is supported to a certain degree by the data in Table 3.8) that the optimal similarity measure depends also upon the nature of the anomaly (i.e. the identity of the masquerading user). This presents a difficulty because, as mentioned in previously, the only data available for training is that of the valid user.

Finally, note that there is a significant disparity between optimal dictionary sizes for the users (1000 vs. 200 sequences). This seems to indicate that USER2's behavior is characterized by a smaller set of actions than is USER0's. The possibility remains, however, that none of the similarity measures investigated here are really appropriate for measuring USER0's behavior, and that under a different measure fewer characteristic sequences would be required. Upon examination, however, we note that instance selection (via the LRU algorithm, as described above) was performed with only 1667 tokens of data for USER0's profile while over 5000 tokens were available for dictionary initialization. When a small number of sequences are rated as highly characteristic by the instance selection algorithm, so few of the other dictionary instances will be touched that selection becomes effectively random for sequences other than the most strongly characteristic ones. This turns out to be the case for USER0 as approximately 260 of the instance selection sequences were devoted to selecting only two of the final dictionary sequences. Thus, we hypothesize that for USER0 the LRU instance selection algorithm concentrates undue attention on sequences that are not necessarily reflective of true behavior while selecting the majority of the instances effectively randomly. Therefore a large number of sequences are required in the final profile to obtain reasonable accuracy. By contrast, LRU seems to select important instances much more successfully for USER2 so fewer are needed in the final profile. Any sequences in the profile beyond those most characteristic of behavior represent noise and lead to decreased performance (and, indeed, degraded performance was seen in the case of USER2's profile for larger final dictionary sizes). The behavior demonstrated here highlights an interaction between the similarity measure and the instance selection algorithm, implying that choice of similarity measure is affected not only by the identity of the profiled user and intruder (and possibly time/concept drift) but also by the choice of instance selection technique.

3.3 Analysis of Concept Drift

When a domain concept is subject to change with time, that domain is said to experience concept drift. In such domains, there is no single, time-independent, optimal hypothesis. Instead, we seek a time-dependent hypothesis that maximizes some criterion (often accuracy) in the total time summation. Here we investigate a process for doing so by attempting to match the instantaneous hypothesis to the instantaneous concept.

We shall begin by introducing some notation. Let $\mathfrak{F}$ be a *feature space* (not necessarily metric) including all possible feature vectors, x, within the domain of interest. Let $\mathfrak{L} = \{L_1, L_2, \ldots, L_N\}$ be the *label space* (alternatively, the *class space*) for the domain, defining all possible class labels for points in $\mathfrak{F}$. Then a *concept* is a partition $C : \mathfrak{F} \to \mathfrak{L}$, assigning a label to each point in $\mathfrak{F}$. We shall write $L_C(x)$ for the label assigned to a particular instance, $x \in \mathfrak{F}$, under concept C. Let us further define the space of *all possible* partitions on $\mathfrak{L}$ to be $\mathfrak{C}$, the *concept space*. The traditional supervised learning problem is then, given a set of feature vectors, $\mathbf{X} \subset \mathfrak{F}$, and the corresponding class labels under concept C, $Y = \{y = L_C(x) : x \in F\}$, formulate a *hypothesis*, $H \in \mathfrak{C}$, such that some value measure (accuracy, classification profit, dollar value, etc.) is maximized, where the value measure, v, is a function of the similarity between the hypothesis, H, and the *true concept*, C. (We ignore, for the moment, the effects of noise which can corrupt measurements of either x or y.)

This is the static case, where C is fixed for the entire course of the learner's lifetime. Now let us consider the addition of time and *concept drift* to the problem. We will consider the time axis to be discrete for this analysis — while this is not necessarily the case, all of our sampling techniques *are* discrete, so this assumption should be sufficient, at least as a first order approximation.

Denote the time steps over which the domain is observed to be

$$t_0, t_1, t_2, \ldots, t_{m-1}, t_m.$$

Assume that at each time step we can sample k points from $\mathfrak{F}$ according to some sampling distribution p_s (we take this distribution to be time invariant, although it need not be so). Furthermore, assume that the true concept is allowed to change with time, so that associated with each time step, t there is a concept, C_t. Now we extend the learning task to formulating time dependent hypotheses, H_t, in an attempt to track the true concept over time such that the total value,

$$v_{\mathrm{tot}} = \sum_{i=0}^{m} v(H_{t_i}, C_{t_i}),$$

is maximized. Initially we observe that a possible approach to this problem is to completely reformulate H at each time step given only the k available samples for that time step. For some domains, this may well be the best possible solution, but we believe that for many domains of real interest it is possible to do better.

We define the *concept drift function*,

$$D : t \to \mathfrak{C},$$

as the mapping of time step to true concept (that is, D defines the change of concept over time). We now propose that a possible approach to handling concept drift is to learn a *base hypothesis*, H_0, and the concept drift function, D, and use knowledge of D to extrapolate $H_1, H_2, \ldots$. Unfortunately, D is not as visible a concept as are the various C_t are, so our first problem is how to measure it.

Define the *conceptual drift set*, $\Delta C_t \subset \mathfrak{F}$, to be the set of points that possess different class labels under concepts C_t and C_{t+1}. That is,

$$\Delta C_t = \{f \in \mathfrak{F} : L_{C_t}(f) \neq L_{C_{t+1}}(f)\}.$$

Now at time step $t+1$, when we wish to revise our theory H_t, we have available to us our new set of instances, $\mathbf{X}_{t+1} = \{x_0, x_1, \ldots, x_{k-1}\}$, sampled from $\mathfrak{F}$ according to p_s. Assuming that all instances are sampled independently (not necessarily a justified assumption), then each one has probability

$$\int_{\Delta C_t} p_s(x)\, dx$$

of being sampled from the drift set, or the expected sampling from the drift set is

$$E(\mathbf{X}) = \int_{\Delta C_t} x p_s(x)\, dx.$$

Thus, in our sample, we should expect to accumulate $|\mathbf{X}|E(\mathbf{X}) = kE(\mathbf{X})$ instances from the drift set. For a sufficiently accurate hypothesis, H_t, these instances should appear as errors (misclassifications) in the application of our hypothesis as a classification model (recall that we are omitting distortion from noise for this analysis). Given knowledge of p_s, we can now estimate $|\Delta C_t|$, which we will call *instantaneous concept velocity at time t*. This functions is an indication of the *amount* by which our hypothesis must be adjusted to reach an acceptable H_{t+1}. More importantly, however, it gives us feedback about the structure of D (or, more specifically, of $|D|$, that is, the functional form of the drift velocity). Proceeding another iteration, we can now approximate $|\Delta C_{t+1}|$

and thence $|\Delta^2 C_t|$. Thus, we have induced a (presumably) differentiable ordinal function on the concept drift space. We can now attempt to model this function by any of a number of methods (solution of the differential equation via numerical methods or by model fitting, learning the functional structure, etc.) Note, however, that lacking any *a priori* information about the form of D, our problem is equivalent to the function induction problem again, and is thus constrained in the same fashion. That is, necessity for bias, a hypothesis language, number of samples required to converge to an accurate model, etc.

3.4 Summary of Current Status

Thus far, we have demonstrated the following points about the application of machine learning to the computer security domain:

- Temporal sequences of commands and command arguments carry user-specific behavioral information sufficient to differentiate users in many cases.
- An instance based learning algorithm (similar to one-nearest-neighbor) can make use of the identifying information available in temporal sequences.
- The closed world assumption (that a user is characterized only by past behaviors and new behaviors represent a different person) can allow learning from positive examples only in this domain.
- The choice of similarity measure, which makes possible generalization under the closed world assumption, has a distinct impact on identification accuracy. Furthermore, the optimal similarity measure may depend not only on the profiled user, but also on the tested (intrusive) user.
- Optimal dictionary size is user dependent.
- Intelligent instance selection is possible, but the choice of instance selection algorithms is user dependent.

Finally, we have made some progress toward a general (domain independent) theory of concept drift.

Chapter 4

Proposed Research

In this chapter, we present the collected set of open issues, and present proposed solutions. On the machine learning side, we need to address learning discrete temporal data and the related problem of learning on spaces without an obvious distance metric, the question of inducing hierarchical models of human problem solving behavior; the question of applicability of hidden Markov models to the anomaly detection task and the underlying issue of HMM structure selection for this domain; and the issue of detecting and dealing with concept drift as it occurs in this domain. On the computer security side, we need to address not only the question of general feasibility of the machine learning methods presented here, but also the question of differentiating 'legitimate' concept drift from that caused by a malicious trusted insider or that caused by hostile training.

4.1 Machine Learning Issues

In this section we describe the major questions that need to be resolved to successfully apply machine learning techniques to the anomaly detection domain and give our initial thoughts on approaches to each question. Our central hypotheses are:

1. Temporal sequences are useful models for this domain and can be induced from user command data.
2. Hidden Markov models can be used to model the class of temporal sequences encountered in this domain. Furthermore, HMM's provide

higher accuracy, faster running times for comparison, and/or smaller space requirements than does our current model of temporal sequences.

3. A hierarchical model of user behavior (as encountered in the anomaly detection domain) can be induced from available data. Furthermore, such a model is useful for the following reasons: accuracy, explanatory power, ability to detect and correct for concept drift, and ability to differentiate concept drift from intrusions and hostile training.

4. Concept drift can be detected and corrected for in this domain, and model independent techniques exist for doing so.

4.1.1 On-line Learning

A fielded anomaly detection system must be on-line; that is, it must accept new data for classification continuously, rather than in batch mode, and must be capable of updating its learned model on the fly. On-line operation is necessary for the investigation of concept drift, but it also has an impact on the design of other components of a learning system. The prototype system presented in Chapter 3 works in batch mode. We will first convert the current system to function in an on-line mode, and will design the components described below to function properly in on-line operation.

4.1.2 Temporal Sequence Learning

Currently, we identify command sequences only as fixed length groupings of specific symbols (Chapter 3). Relative importance of sequences is measured only in terms of age and of frequency that individual sequences are measured as being 'most similar' to independent 'parameter selection' sequences. We propose to investigate methods for identifying and extracting more complex representations of sequences (allowing, for example, 'don't care' states or regular expression-like representations). Sequence matching algorithms based on minimum edit distance [15] can recognize arbitrary gaps in sequence matches, equivalent to the $*$ regular expression construct. Hidden Markov models are also attractive in this context, as they can represent arbitrary regular languages.

We propose to develop alternative similarity measures based on edit distance measures and on hidden Markov models. Edit distance based measures

```
Until a single observation symbol remains, repeat:

  1. Group all sequences into sequence clusters (representing
     classes of behaviors) and 'outliers' (representing actions
     that fit no known class of behaviors)

  2. Replace all class members (sequences of symbols) with
     their class label

Return the new observation sequence as a user profile
```

Figure 4.1: Proposed behavioral hierarchy construction algorithm.

would compare incoming sequences to historical 'template' patterns stored in the user profile, much as the current similarity measures do. The principle difference is that edit distance would allow non-fixed-length patterns and would allow gaps in matched patterns. We describe HMM based similarity measures further in Section 4.1.4.

4.1.3 Induction of a Hierarchical Model of Behavior

We propose to model human-computer interactions as a hierarchy of increasingly abstract 'behavior clusters'. There are usually multiple ways to accomplish a given task with a computer, and one must be selected for each occurrence of the task. Rather than formulating a novel solution each time a task is encountered, humans tend to develop repeated patterns for accomplishing frequently encountered tasks. These fixed patterns, in turn, form building blocks for yet more complex behaviors. For example, in a UNIX context, a low level behavioral pattern might be the command sequences a user employs to navigate and search the directory tree while a higher level behavioral pattern might be programming — a task that employs directory navigation as a subtask. Thus, we expect a large fraction of a user's behavior to be representable in terms of successively more complex behavior patterns, where each level of the hierarchy is depends only on lower levels.

Our initial approach to hierarchy construction is described by the algorithm in Figure 4.1. This process iteratively accumulates disjoint sequences of symbols from the observation stream into sequence classes. Sequences can be grouped via a clustering algorithm that attempts to minimize net inter-cluster similarity and maximize net intra-cluster similarity. Sequences that do not match any cluster sufficiently well are retained as 'outliers' and not assigned a class label. Each non-outlier sequence of symbols is then replaced with its class label, thereby reducing the total number of symbols in the observation sequence. At the next iteration of the accumulation step, the class labels function just as ordinary observation sequences, and sequence classes created at this step express the relations between the behavioral groupings created at previous steps. When only a single token remains or no further sequence groupings can be made, the process is halted and the resulting structure is returned.

A hierarchical model of behavior has the following advantages:

Model Efficiency Conceivably, a user's behavioral patterns could be modeled as a single, extremely complex, network of atomic actions interconnected as an arbitrary graph. While appealing for its ability to model an broad set of behaviors, it is likely that such a model cannot be practically induced from real data. For an alphabet, Σ, of $|\Sigma|$ symbols with only first-order relations (i.e. only pairwise relations between symbols are considered), there are $|\Sigma|^2$ possible interconnections for such a network, and, thus, $|\Sigma|^2$ parameters to select. Furthermore, first order relations may not be sufficient to model the types of context information necessary for accurate behavioral representations. For example, in the context of programming, the command `vi` might be most likely followed by the command `gcc` while in the context of writing a PhD prelim, the command `vi` might be most likely to be followed by the command `latex`. But extension of the arbitrary graph model to include contexts via higher order relations introduces an exponential complexity growth.

A hierarchical model possesses at most $|\Sigma|$ interconnections (each node having only a single parent). The alphabet is drawn from quite a different set of symbols. The nodes of the hierarchical model we are proposing are short sequences of events, each sequence possibly being equivalent to a regular expression of fixed (short) length. Thus, each sequence of length k possesses k^2 possible interconnections. The entire hierarchy thus has k^2N interconnections, where N is the number of 'important' sequences. It is to be hoped that $N << |\Sigma|^2$. The hierarchical model, therefore, approximates the full explicit model with a set of locally constructed patches and a hierarchical interconnection of those patches.

Explanatory Power In many domains, detection of anomalies is most useful when they can be explained. In a computer security context, each detected anomalous situation must be reviewed by a human and appropriate action taken. In such circumstances, a humanly comprehensible explanation for the detection can be quite useful. For example, an explanation of a detection by a first order statistical detector might be: "This activity is suspicious because it involved 1.2 standard deviations fewer occurrences of the command `ls` than expected"; while explanations constructed from a hierarchical model might read: "This activity is suspicious because it deviates from the 'code development' behavioral pattern, while maintaining the expected distribution of all atomic actions" or "This activity is suspicious because it does not match any known behavioral pattern, yet maintains the expected distribution of atomic events and macros." The increased explanatory power of a hierarchical model is derived from its attempt to induce the structure of the user's problem solving hierarchy. To the extent that this effort is successful, we obtain not only knowledge of the user's behavioral patterns, but also of the user's typical tasks, work habits, and problem solving strategies.

Addressing the Hostile Training and Trusted Insider Problems It is also possible to formulate approaches to the hostile training and trusted insider problems of computer security in terms of a hierarchical behavioral model. While we defer the full discussion of this aspect to Section 4.2, we give a short summary here. We hypothesize that hostile training and insider attacks will appear to the system as characteristic behavioral deviations at certain levels of the hierarchy. Deviations at the lowest level of the hierarchy may indicate only learning or change on the part of the valid user; deviations at the middle levels of the hierarchy may indicate changing tasks (potentially to malicious or hostile tasks) on the part of the valid user; and deviations at the highest levels may indicate a knowledgeable hostile trainer. Evaluation of our hypotheses will require data sets representing these types of attacks. If we are unable to locate real data traces of these attacks, we intend to synthesize these attacks as a baseline evaluation of our hypotheses.

4.1.4 Hidden Markov Models

We propose to investigate hidden Markov models as user behavioral models. An HMM can capture aspects of both deterministic and stochastic user behaviors. The finite state transition matrix can describe the deterministic process by which a certain task is accomplished, while transition probabilities allow flexibility in the process or allow one procedure to be preempted in favor of another. The stochastic output processes can account for noise or multiple equivalent methods for accomplishing a single step of a task (for example, `awk`, `sed`, and `perl` can all be used to accomplish the same batch mode edit).

For a given class of behaviors, an HMM can be constructed to represent that class. A user profile will then be a dictionary (or, possibly, a hierarchy) or HHM's for behaviors. New incoming observations can be matched against each model in the dictionary via the Forward-Backward algorithm, [50], and the newly observed sequence can be labeled according to the closest match model (i.e. the HMM with the highest probability of having generated that observation sequence). Intuitively, this is similar to the application of HMM's to the continuous speech recognition problem, where HMM's can model the similarity of phonemes or words to speech observed during training.

One of the principle difficulties in the use of HMM's as time sequence models is structure selection (sometimes referred to as model or topology selection). It's apparent that that the HMM parameter set discussed in Appendix A omits the values N (number of states), M (output alphabet size), and $\mathbf{V}$ (output alphabet symbols), though M and $\mathbf{V}$ can be reasonably estimated either from O or from the known set of atomic events. Furthermore, although zero transition probabilities are encoded in A, they are not found by the Baum-Welch training algorithm. We denote the set of such zero probability transitions as $\mathbf{Z} = \{z_{ij} = (i,j) | a_{ij} = 0\}$. Then the *structure* of the HMM is the set ⁂ $= \{N, M, \mathbf{V}, Z\}$.

Smyth et al. [63] indicate that it is, in principle, possible to solve for an optimal structure under a prior distribution on the space of possible structures, $p($⁂$)$. They describe the technique of Bayesian model averaging, which constructs a mixture model whose predictions take into account the predictions of all possible structures weighted by their prior probability. Unfortunately, for most systems of interest this calculation is infeasible. Alternatives include estimates of the Bayesian average model such as cross validation, hypothesis testing, and a Monte Carlo estimates.

Alternatively, domain knowledge can be used to select the model structure. This approach is promising when, for example, the underlying states and processes driving the observed process are known. Denning's proposal of mapping audit events to Markov state variables is an example of this case [17].

Successful application of HMM's to the anomaly detection domain will require determination of ∴ that are appropriate for this domain. While the general solution to the HMM structure selection problem is beyond the scope of this research, a specific solution for the anomaly detection domain is reasonable to undertake. Appropriate structures may may be selected by hand from known properties of the domain, or may be inferred automatically from observations. Known properties include the grammatical structure of the command line interface and semantic relations between commands (a compiler produces an executable which may appear later as a command, for example). We will initially investigate hand-built HMM structures.

Domain knowledge can also be used to improve automated structure induction. Knowledge of, for example, grammatical structure of the command line environment may allow us to constrain the space of possible models to one that can be analyzed via Bayesian model averaging. A more general structure inference algorithm (one not dependent on domain knowledge) would examine an event stream and derive an acceptable structure directly from observed properties of that stream. For example, cycles in the HMM structure should correspond to peaks in the autocorrelation function of the event stream. Accumulation of such observations may allow selection of a particular structure or may narrow the space of possible structures to a more manageable one. We propose to investigate semiautomatic (employing domain knowledge) and automatic methods for inducing ∴. We also intend to investigate whether optimal HMM structures are user dependent or are universal.

Once an appropriate HMM structure has been selected, we can compare the performance of HMM's to that of our established techniques to determine the general applicability of HMM's as detectors for the anomaly detection domain. Our intent is to use HMM's as sequence based similarity measures, similar to the instance based technique we have used for our current results (see Chapter 3). HMM's can be trained as recognizers for command sequences and the user profile can be formed out of a set of such sequence models. Anomaly detection would proceed much as currently implemented, with the observation probability rule (see Appendix A) replacing the current similarity measure. We can then test the utility of the HMM model against the instance based model for accuracy and time/space efficiency. If HMM's prove valuable in this context, we may attempt a similar integration with the hierarchical behavior model structure proposed above.

4.1.5 Concept Drift

We propose to analyze the theoretical properties of concept drift and the practical application to the specific formulation encountered in the anomaly detection domain. In this section, we examine the theoretical and practical formulations of concept drift in this domain.

Theoretical Analysis

We propose to continue the theoretical analysis of concept drift presented in Section 3.3. In particular, restriction to metric feature spaces, handling noise, and restriction to cover extant techniques (such as instance pruning, forgetting, knowledge transfer, etc.) are needed. The most pressing issue, however, is the need for methods for adapting current models in the face of drift. We conjecture that the update rule for hypotheses is learning algorithm specific, but that by further constraining the nature of the feature space, $\mathfrak{F}$, (requiring it to be metric, for example) we will be able to obtain more information about the drift function, D, such as regions of high and low drift velocity or possibly an n dimensional 'conceptual moment of inertia'.

Practical Implementation

Although it is generally more desirable to begin with a complete theory of a phenomenon and to derive a specific application from that, our current level of development in the theoretical branch of concept drift is not sufficient to derive a working model. Therefore, we propose an empirically derived, domain specific approach for use in the anomaly detection task so that we can work concurrently on practice and theory. As the theory is developed, we may reformulate the current approach.

One domain-specific concept drift issue is that of differentiating 'legitimate' concept drift (drift caused by normal changes on the user's part) from anomalies caused by an intruder or arising from abusive actions on the part of the valid user. All such activities may appear anomalous to the detection system, yet must be handled quite differently. Specifically, we categorize the possible sources of anomalies as follows:

1. The user may learn how to accomplish an old task in a new manner, thus partially or completely replacing previous behavior patterns for accomplishing that task but maintaining a similar context for those patterns.

2. The user may undertake a new task which can be solved with familiar techniques. Here we expect to see familiar behavior patterns in novel contexts.

3. The user may undertake new tasks in new ways.

4. An intruder may gain entrance to the user's account, without attempting to emulate the valid user's behavior patterns.

5. An intruder may gain access to the user's account and attempt to emulate the user's behaviors.

The last case cited may not even appear anomalous to the detector, depending on the talent of the intruder. In most cases of interest (i.e. when the intruder has hostile intent), however, we expect that either the intruder will undertake an abusive action and appear in category two or four, or the intruder will attempt to train the user profile into accepting new behaviors as normal. We will discuss implications of the various intruder scenarios and the abusive insider problem in the next section, concentrating here on methods for handling the first three cases.

We base our approach to concept drift on the hierarchical model of sequence learning presented previously. We hypothesize that each class of drift will appear as a deviation from expected behavior at a different level of the hierarchical model. In the first case above, for example, the system would see previously unknown commands or sequences of commands. While such a difference could be immediately flagged as an anomaly, the existence of a surrounding model may allow the new phenomenon to be positioned within the context of a higher level task or behavior. If the user is changing low level behaviors (say, learning a new command or introducing a new alias) only incrementally and relatively slowly, then each new command should occur within a context of familiar commands.

For the second case, we expect to see only familiar commands and command sequences. A detector based on atomic events or one based on sequences of events would detect no anomaly in this case, yet change *is* occurring. To a hierarchy-based detector, the change is apparent in anomalies at middle or upper levels of the hierarchy, while none are detected in the lower tiers.

The third case should evidence anomalies at many levels of the hierarchy. In this case, the activity may bear high resemblance to the fourth case, and may have to be treated as an intrusion. But if, as above, the user is changing behaviors only incrementally and slowly, it may be possible to recognize the new behaviors within contexts and adapt to them.

Thus, we hypothesize that concept drift can be detected through the appearance of behavioral anomalies with respect to the current model. When the anomalies can be classified as benign (when they fall definitely into categories one or two, for example), then the new data can be incorporated into the user model. When the anomaly is known to fall into category four or five, or cannot be classified with confidence, then the system should forward the activity to a human supervisor for feedback and appropriate action. The features that allow us to detect and properly handle anomalous activities, therefore, are:

- The observation of deviations from expected behavioral patterns.
- The tier(s) of the hierarchy at which those deviations occur.
- The rate of behavioral change.

Effectively, change higher in the hierarchy indicates more drastic behavioral changes, as does high rate of change. Because such drastic changes are more likely to be dangerous, they should be investigated by the user, a security officer, or a system administrator.

4.2 Computer Security Issues

While most of the previously presented material has direct bearing on the computer security aspects of the anomaly detection domain, two issues have little direct bearing on the machine learning facets: the trusted insider and the hostile training problems. Our hypotheses on these issues can be summarized as:

- A trusted insider undertaking abusive actions should appear to the anomaly detector as a characteristic type of drift within the hierarchical behavioral model previously proposed. Such drift may be similar to that caused by certain types of normal change, but the incidence of this type of change may be low enough to make it practical to analyze individual incidents by hand.
- Hostile training by a sufficiently knowledgeable intruder may well be beyond the abilities of this type of detector to identify. Nevertheless,

there are at least three possible factors that can mitigate the hazards of this particular attack:

- The intruder may act quickly enough to become apparent as either an intruder or as an abusive trusted insider and be detected.
- The valid user's behavior may be baselined and change that drifts too far from the baseline could be flagged for further investigation.
- A formal characterization of concept drift may allow a user's characteristic change pattern to be learned as well as the user's characteristic behaviors. Then change which deviates from the expected drift patterns could be flagged as illegal.

4.2.1 The Trusted Insider Problem

Not all abusive actions on the part of a trusted insider may be detectable by the anomaly detection system. Some actions may be abusive only when seen within a larger context which is invisible to the anomaly detector. Nonetheless, there are some scenarios that may fall within the domain of anomaly as defined by the detection system we propose. Specifically, when the system abuse falls outside the user's normal behavioral patterns, it should be noticed by the anomaly detector. We hypothesize that such occurrences will fall into either category two or three (as defined in Section 4.1.5) and that they will appear as drift at particular levels of the hierarchical behavior model (Sections 1.2.2 and 4.1.3). We propose to examine the characteristics of concept drift introduced by a hostile insider in terms of these categories and the hierarchical behavior model.

Data representing abusive trusted insiders will be required to evaluate our hypothesis about the characteristics of abusive insider behavior patterns. If we are unable to locate data traces of real instances of trusted insider attacks, we intend to synthesize such data.

4.2.2 The Hostile Training Problem

A solution to the general hostile training problem is likely to be beyond the scope of the anomaly detection techniques proposed in this document. Particular special cases, however, may be detectable by the system. We intend to investigate the abilities of the system proposed here to handle a hostile training scenario.

If a hostile teacher acts too quickly, the goal behaviors may become apparent in class two of the concept drift categories introduced in Section 4.1.5, and as deviations from expected behaviors at the corresponding level of the hierarchical behavior model. In this case, the behavior could be flagged for further investigation by a software or human agent. Alternatively, the user's profile could be baselined at some point and any change that drifts too far from the baseline (in terms of, for example, degree of change within the behavior hierarchy or number of new HMM's introduced as pattern detectors) could be flagged for further investigation. The utility of either of these solutions would depend on the policies of the site in question and how they are willing to make the tradeoff between detection accuracy and human inconvenience.

A third possibility is that a user's behaviors may change in characteristic and predictable ways (i.e. we may be able to learn an estimate of the D function described in Section 3.3), and that a hostile teacher may violate these expected drift patterns. For example, we hypothesize that hostile training will appear as gradual introduction of new low-level structures (commands and command patterns) during the training phase followed by an abrupt shift in the high level structures mostly employing the new low level patterns (i.e. new tasks employing the newly learned commands and patterns) during the attack phase. This type of behavioral change is hopefully not characteristic of many users, and may serve as a distinguishing feature. Thus, a user could be characterized not only by known behaviors but also by the ways in which those behaviors are characteristically modified. An intruder attempting to train the system would then have to not only emulate the user's behaviors but also the behavioral drift function. Hopefully, this is a more difficult task.

To examine characteristics of hostile training and the capacity of our anomaly detection system to notice it, data representing hostile training sessions will be needed. If no real data traces can be located, we intend to synthesize such data.

4.3 Summary and Prioritization

We acknowledge that a complete characterization of the anomaly detection domain and thorough exploration of even the aspects presented here is beyond the scope of this limited research project. We close, therefore, with a prioritization of the topics we have presented. This represents our view of which topics are most interesting and important, and which will be undertaken if time and resources permit.

1. Construction and testing of a hierarchical model of use behavior in terms of tiers of sequences, the lowest level sequences being comprised of atomic events.

2. Methods for hidden Markov model structure selection for the anomaly detection domain.

3. Methods for detecting and adapting to concept drift, both in practice (in terms of the behavior hierarchy, should it prove to be useful) and in theory.

4. Methods for representing and extracting more complex types of temporal sequence relations.

5. Evaluation of all techniques on real user data and empirical comparison to standard techniques from the machine learning and computer security literature.

6. Collection of more data from more disparate sources (users with different experience bases, abusive trusted insiders, hostile training episodes, etc.), including synthesized data for otherwise unavailable cases.

7. Testing of the anomaly detection system against the trusted insider and hostile training attacks.

8. Testing of the anomaly detection system against other event types and sources (such as DOS command prompts, GUI event streams, audit log data, network traffic, or system call traces).

References

[1] D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. In machine learning journal [36], pages 37–66.

[2] D. W. Aha. Incremental constructive induction: An instance-based approach. In L. A. Birnbaum and G. C. Collins, editors, *Machine Learning: Proceedings of the Eighth International Workshop*, pages 117–121, Evanston, IL, 1991. Morgan Kaufmann.

[3] D. Anderson, T. Frivold, and A. Valdes. Next-generation Intrusion-Detection Expert System (NIDES). Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, CA, May 1995.

[4] D. Anderson, T. Lunt, H. Javitz, A. Tamaru, and A. Valdes. Safeguard final report: Detecting unusual program behavior using the NIDES statistical component. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, Dec 1993.

[5] J. P. Anderson. Computer security threat monitoring and surveillance. Technical Report Technical Report, Washington, PA, 1980.

[6] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in statistical analysis of probabilistic functions in Markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[7] J. Baxter. A Bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine Learning*, 28(1):7–39, Jul 1997.

[8] P. Boedges. Air force mounts offensive against computer crime. Government Computer News, Jul 1988.

[9] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997.

[10] R. Caruna. Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning: Proceedings of the Tenth International Conference*, pages 41–48, Amherst, MA, 1993. Morgan Kaufmann.

[11] R. Caruna, S. Baluja, and T. Mitchell. Using the future to 'sort out' the present: Rankprop and multitask learning for medical risk evaluation. In Touretzky et al. [72].

[12] T. Chenoweth and Z. Obradovic. An explicit feature selection strategy for predictive models of the S&P 500 index. *NeuroVe$t Journal*, 3(6):14–21, 1995.

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

[14] S. Cheung, K. N. Levitt, and C. Ko. Intrusion detection for network infrastructures. In *The 1995 IEEE Symposium on Security and Privacy*, May 1995. Short Presentation.

[15] Thomas HOA. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. The MIT Press, Cambridge, MA, 1992.

[16] J. Denker, D. Schwartz, B. Wittner, S. Solla, J. Hopfield, R. Howard, and L. Jackel. Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1:877–922, 1987.

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

[18] D. Farmer and W. Venema. SATAN overview (Security Administrator Tool for Analyzing Networks). Electronic release, Mar 1995. Program documentation for the SATAN/SANTA tool.

[19] S. Forrest, S. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 1996.

[20] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *Proceedings of 1996 IEEE Symposium on Computer Security and Privacy*, 1996.

[21] K. Fukunaga. *Statistical Pattern Recognition (second edition)*. Academic Press, San Diego, CA, 1990.

[22] S. Gordon. Current computer virus threats, countermeasures, and strategic solutions. White paper, McAfee Associates, 1996.

[23] S. J. Hanson, C. L. Giles, and J. D. Cowan, editors. *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, 1993.

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

[25] W. Hunteman. Automated information system—(ais) alarm system. In *Proceedings of the Twentieth National Information Systems Security Conference* [40], pages 394–405.

[26] H. Javitz and A. Valdes. The SRI IDES statistical anomaly detector. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 316–326, 1991.

[27] I. Krsul, E. Spafford, and T. Tuglular. A new approach to the specification of general computer security policies. Technical Report COAST TR-97-13, Purdue University, West Lafayette, Indiana, 1997. Submitted to the 1998 IEEE Symposium on Security and Privacy.

[28] S. Kumar. *Classification and detection of computer intrusions*. PhD thesis, Purdue University, W. Lafayette, IN, 1995.

[29] S. Kumar and E. Spafford. An application of pattern matching in intrusion detection. Technical Report CSD-TR-94-013, Purdue University, West Lafayette, Indiana, Jun 1994.

[30] T. Lane and C. E. Brodley. An application of machine learning to anomaly detection. In *National Information Systems Security Conference*, Baltimore, MD., 1997.

[31] T. Lane and C. E. Brodley. Detecting the abnormal: Machine learning in computer security. Technical Report TR-ECE 97-1, Purdue University, School of Electrical and Computer Engineering, West Lafayette, IN, 1997.

[32] T. Lane and C. E. Brodley. 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*, 1997.

[33] D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 148–156, New Brunswick, NJ, 1994. Morgan Kaufmann.

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

[35] T. F. Lunt and R. Jagannathan. A prototype real-time intrusion-detection expert system. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 59–66, 1988.

[36] *Machine Learning.*

[37] C. D. Mitchell. *Improving Hidden Markov Models for Speech Recognition.* PhD thesis, Purdue University, W. Lafayette, Indiana, May 1995.

[38] T. M. Mitchell. The need for biases in learning generalizations. Technical Report Report CBM-TR-117, Rutgers University, New Brunswick, NJ, 1980.

[39] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, 1994.

[40] The National Institute of Standards and Technology and the National Computer Security Center. *Proceedings of the Twentieth National Information Systems Security Conference*, 1997.

[41] S. W. Norton. 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*, pages 657–663, Seattle, WA, 1994.

[42] A. Oppenheim and R. Schafer. *Discrete-Time Signal Processing.* Signal Processing. Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[43] P. Porras and P. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the Twentieth National Information Systems Security Conference* [40], pages 353–365.

[44] L. Pratt. Discriminability-based transfer between neural networks. In Hanson et al. [23], pages 204–211.

[45] L. Pratt. Transfer between neural networks to speed up learning. *Journal of Artificial Intelligence Research*, submitted 1996.

[46] L. Pratt, J. Mostow, and C. Kamm. Direct transfer of learned information among neural networks. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA, 1991. MIT Press.

[47] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[48] J. R. Quinlan. Learning logical definitions from relations. In machine learning journal [36], pages 239–266.

[49] J. R. Quinlan. *C4.5: Programs for machine learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[50] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, February 1989.

[51] M. Ring. Learning sequential tasks by incrementally adding higher orders. In Hanson et al. [23], pages 115–122.

[52] M. Ring. *Continual Learning in Reinforcement Environments.* PhD thesis, University of Texas at Austin, Austin, Texas 78712, Aug 1994.

[53] M. Ring. CHILD: A first step towards continual learning. *Machine Learning*, 28(1):77–104, Jul 1997.

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

[55] J. C. Schlimmer. *Concept acquisition through representational adjustment.* PhD thesis, University of California, Irvine, 1987.

[56] J. Schmidhuber. On learning how to learn learning strategies. Technical Report FKI-198-94, Technische Universität München, 80290 München, Germany, 1994.

[57] J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28(1):105–130, Jul 1997.

[58] M. Sebring, E. Shellhouse, M. Hanna, and R. Whitehurst. Expert systems in intrusion detection: A case study. In *Proceedings of the 11th National Computer Security Conference*, pages 74–81, Oct 1988.

[59] S. E. Smaha. Haystack: An intrusion detection system. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pages 37–44, 1988.

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

[61] P. Smyth. Markov monitoring with unknown states. *IEEE Journal on Selected Areas in Communications, special issue on intelligent signal processing for communications*, 12(9):1600–1612, 1994.

[62] P. Smyth. Clustering sequences with hidden Markov models. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing 9*. MIT Press, 1997.

[63] P. Smyth, D. Heckerman, and M. Jordan. Probabilistic independence networks for hidden Markov models. *Neural Computation*, 9, in press.

[64] E. Spafford. The Internet Worm program: An analysis. *ACM Computer Communication Review*, 19(1):17–57, Jan 1989.

[65] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – a graph-based intrusion detection system for large networks. In *Proceedings of The 19th National Information Systems Security Conference*. The National Institute of Standards and Technology and the National Computer Security Center, Oct 1996.

[66] C. Stoll. *The Cuckoo's Egg*. Pocket Books, 1989.

[67] H. S. Teng, K. Chen, and S. C. Lu. Adaptive real-time anomaly detection using inductively generated sequential patterns. In *Proceedings of the IEEE Computer Society Symposium on Research in Computer Security and Privacy*, pages 278–284, Los Alamitos, CA, 1990. IEEE Computer Society, IEEE Computer Society Press.

[68] H. S. Teng, K. Chen, and S. C. Lu. Security audit trail analysis using inductively generated predictive rules. In *Proceedings of the 6th Conference on Artificial Intelligence Applications*, pages 24–29. IEEE, IEEE Service Center, Piscataway, NJ, Mar 1990.

[69] S. Thrun. Lifelong learning: A case study. Technical Report CMU-CS-95-208, School of Computer Science, Carnagie Mellon University, Nov 1995.

[70] S. Thrun. Is learning the n-th thing any easier than learning the first? In Touretzky et al. [72].

[71] S. Thrun and J. O'Sullivan. Discovering structure in multiple learning tasks: The TC algorithm. In L. Saitta, editor, *Proceedings of the Thirteenth International Conference on Machine Learning*, San Mateo, CA, 1996. Morgan Kaufmann.

[72] D. Touretzky, M. Mozer, and M. Hasselmo, editors. Cambridge, MA, 1996. MIT Press.

[73] R. Violino. The security facade. InformationWeek, Oct 1996.

Appendix A

An Overview of Hidden Markov Models

In the following discussion, we adopt the notation and general framework presented in [50][1]. An HMM consists of four components:

1. A set of N states denoted $\mathbf{Q} = \{q_1, q_2, \ldots, q_N\}$. The state of the system at time t is denoted s_t.

2. A state transition probability matrix, A, of the form:

$$\mathsf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix}$$

   where a_{ij} denotes the probability of transitioning from state q_i to state q_j. That is, the transition probabilities express the relationship: $a_{ij} = P(s_{t+1} = q_j | s_t = q_i)$. A transition probability of $a_{ij} = 0$ indicates that state q_j cannot be reached from state q_i in a single time step. Because of the nature of the training algorithms, zero probability transitions must be initialized as such before training. Thus, the graphical structure of the network must be known *a priori*.

[1]HMMs can also be described as special cases of Bayesian belief networks, though we omit that characterization here. For such an analysis, see [63].

3. An output (or observables) distribution for each state, $\mathbf{B} = \{b_i\}$. For the case of discrete observations (as encountered in the anomaly detection domain, where observations correspond to commands), the output distributions are characterized by a discrete alphabet of M symbols denoted $\mathbf{V} = \{v_1, v_2, \ldots, v_M\}$ and a discrete probability distribution, b_j on that alphabet. The distribution b_j gives the probability of observing symbol v_k at time t when the system is in state j. That is, $b_j(k) = P(v_k \text{ observed } | s_t = q_j)$.

4. An initial state distribution, $\pi = \{\pi_i = P(s_1 = q_i)\}$, expressing the probability that the system starts in state q_i.

By the *parameter set* of the model, we mean the set $\lambda = \{\mathsf{A}, \mathbf{B}, \pi\}$. There are a number of notable properties associated with this definition of HMMs. First, the transition probabilities are independent of time, as are the output distributions. Thus, the process described by the HMM is stationary. Second, the HMM has a finite memory (a single state, for this case, although more general cases with deeper memories are possible). This is the Markov property, and allows calculation of marginal probabilities without exhaustive enumeration of all state sequences (that is, $P(s_t = q_i | s_{t-1} = q_{i'}, s_{t-2} = q_{i''}, \ldots, s_1 = q_{i^{(t-1)}}) = P(s_t = q_i | s_{t-1} = q_{i'})$).

There are three fundamental problems associated with the practical implementation of HMMs as prediction or estimation models:

Observation Probabilities: Given a sequence of observations, $O = O_1, O_2, \ldots, O_T$, and a model, λ, calculate the probability of observing that sequence of observations under that model, $P(O|\lambda)$.

State Sequence Selection: Given a sequence of observations, $O = O_1, O_2, \ldots, O_T$, and a model, λ, calculate the sequence of states, $Q = q_1, q_2, \ldots, q_T$, most likely (under some optimality criterion) to have generated O.

Model Training: Given a sequence of observations, $O = O_1, O_2, \ldots, O_T$, select the parameter set, $\lambda = (\mathsf{A}, \mathbf{B}, \pi)$, that maximizes $P(O|\lambda)$.

We present brief descriptions of the commonly employed techniques for the solutions of these problems here.

Calculation of Observation Probabilities The first problem is solved with the *forward-backward algorithm* (F-B). This is a dynamic programming algorithm that employs the Markov property (finite memory) to avoid computation of all N^T possible state sequences of length T. The forward step is sufficient to calculate observation probabilities, so we defer discussion of the backward step until later.

The forward step calculates successive probabilities of partial observation sequences $\alpha_t(i) = P(O_1, O_2, \ldots, O_t, s_t = q_i|\lambda)$ (the probability that model λ is in state q_i, having seen observations $O_1, O_2, \ldots, O_t$). The partial sequence probabilities are initialized to:

$$\alpha_1(i) = \pi_i b_i(O_1) \qquad 1 \leq i \leq N.$$

Then the partial sequences are incrementally extended according to the rule:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}).$$

This is the point at which the Markov property is employed. Lacking a guaranteed finite memory, the sum over $\alpha_t(i)$ would explode to $\alpha_t(s_t = q_i, s_{t-1} = q_{i'}, s_{t-2} = q_{i''}, \ldots, s_1 = q_{i^{(t-1)}})$. Finally, when the partial sequences have been extended to $t = T$, the total probability of observation sequence O can be calculated as:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

The probability of observation sequence O under model λ can, therefore, be calculated in $\mathbf{O}(N^2T)$ steps.

Determination of Optimal State Sequences For most HMMs of interest, there are a variety of different state sequences that can produce the same observation sequence (possibly even with similar or even equal probabilities). Thus, it's impractical or even impossible to identify a single 'correct' sequence state corresponding to a given observation sequence. It is, however, possible to introduce an optimality criterion and to select the optimal state sequence under that assumption. While a number of such criteria are possible, a commonly used one is to select the state sequence that maximizes $P(Q|O, \lambda)$ (or, equivalently, maximize $P(Q, O|\lambda)$). This is the optimality criterion employed in the *Viterbi algorithm*. The Viterbi algorithm employs dynamic programming in a fashion similar to the F-B algorithm, differing mainly in that it calculates a maximum probability among state paths, rather than a total probability across all such paths. Also, an auxiliary array is kept so that the optimal state path may be retrieved at termination.

Analogously to the partial sequence probabilities, $\alpha_t(i)$ employed by the F-B algorithm, the Viterbi algorithm uses the quantity

$$\delta_t(i) = \max_{s_1, s_2, \ldots, s_{t-1}} P(s_1, s_2, \ldots, s_t = q_i, O_1, O_2, \ldots, O_t | \lambda)$$

which is the value (probability) of the most likely path of length t that ends in state q_i and accounts for the first t observations. The array, $\psi_t(i)$, is kept simultaneously to track the state that maximized the path value ending in state q_i at time t. The arrays are initialized to:

$$\begin{aligned} &\delta_1(i) = \pi_i b_i(O_1), && 1 \leq i \leq N \\ &\psi_1(i) = 0, && 1 \leq i \leq N. \end{aligned}$$

The δ paths are incrementally extended and the ψ array is updated to track the best path passing through state j at time t by:

$$\begin{aligned} &\delta_t(j) = \max_{1 \leq i \leq N} \{\delta_{t-1}(i) a_{ij}\} b_j(O_t), && 1 \leq j \leq N \\ &\psi_t(j) = \operatorname*{argmax}_{1 \leq i \leq N} \{\delta_{t-1}(i) a_{ij}\}, && 1 \leq j \leq N. \end{aligned}$$

When the paths have been extended to $t = T$, the final path probability and most likely state path are retrieved by:

$$\begin{aligned} &P^* = \max_{1 \leq i \leq N} \{\delta_T i\} \\ &q_T^* = \operatorname*{argmax}_{1 \leq i \leq N} \{\delta_T(i)\} \\ &q_t^* = \psi_{t+1}(q_{t+1}^*), && t = T-1, T-2, \ldots, 1. \end{aligned}$$

Again, this calculation requires $\mathbf{O}(N^2 T)$ steps.

Training the Model To select the parameter set, $\lambda = (\mathsf{A}, \mathbf{B}, \pi)$ such that the probability of a given observation sequence, O, is maximized, we employ the *Baum-Welch* (B-W) algorithm. B-W is a case of the EM (estimation-maximization) method and fundamentally works by gradient descent on the $P(O|\lambda)$ space. There are two important consequences of this approach. First, we can only expect to converge to a locally optimal solution for the value of λ, and, second, the value to which we initialize λ is likely to have a great impact on the final solution.

For the B-W algorithm, we introduce the backward partial sequence variables, $\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T | s_t = q_i, \lambda)$, analogous to the forward variables $\alpha_i(t)$. The backward variables encode the probability of observing the partial sequence beginning with O_{t+1} and continuing to time T, and are calculated in a manner similar to the forward partial sequence variables. Together, the calculation of the forward and backward variables constitutes the entire F-B algorithm.

With both α's and β's in hand, we can define the observed state transition probability variable, $\xi_t(i,j)$ as:

$$\begin{aligned}
\xi_t(i,j) &= P(s_t = q_i, s_{t+1} = q_j | O, \lambda) \\
&= \frac{P(s_T = q_i, s_{t+1} = q_j, O | \lambda)}{P(O|\lambda)} \\
&= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \\
&= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\displaystyle\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}.
\end{aligned}$$

Thus, $\xi_t(i,j)$ is an *a posteriori* estimate of a_{ij}; it represents the probability that the model is in state q_i at time t and in state q_j at time $t+1$ given the observation sequence O. Now we define $\gamma_t(i)$, the probability that the model is in state q_i at time t as:

$$\begin{aligned}
\gamma_t(i) &= P(s_t = q_i | O, \lambda) \\
&= \textstyle\sum_{j=1}^{N} \xi_t(i,j).
\end{aligned}$$

Then the sum of $\gamma_t(i)$ over all $t = 1 \ldots T-1$ is the expected number of times that a transition takes place *away* from state q_i. Similarly, the sum of $\xi_t(i,j)$ from 1 to $T-1$ expresses the expected number of times that the transition $q_i \to q_j$ is made. Now we can write a set of reestimation formulae for the parameters π_i, a_{ij}, and $b_j(k)$:

$$\begin{aligned}
\overline{\pi_i} &= \gamma_1(i) \\
\overline{a_{ij}} &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\
\overline{b_i(k)} &= \frac{\sum_{\substack{t=1 \\ O_t=v_k}}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}
\end{aligned}$$

We denote the reestimated parameter set $\overline{\lambda} = (\overline{\mathsf{A}}, \overline{\mathbf{B}}, \overline{\pi})$. Rabiner notes that in [6], Baum et al. have showed that $P(O|\overline{\lambda}) \geq P(O|\lambda)$ and, therefore, the iterative process of replacing λ with $\overline{\lambda}$ converges to a local maximum on the probability space $P(O|\lambda)$.

An important feature of the parameter reestimation procedure given here is that the term a_{ij} appears in the numerator of the calculation of $\xi_t(i,j)$ and, therefore, of $\overline{a_{ij}}$. Thus, a zero probability transition will never be updated to a non-zero probability by the B-W algorithm. Furthermore, the updated transition probability, $\overline{a_{ij}}$, can be zero only if $a_{ij} = 0$, $\alpha_t(i)\beta_{t+1}(j) = 0\ \forall t$ (i.e. no path ever includes the transition $q_i \rightarrow q_j$), or $b_j(O_{t+1}) = 0\ \forall t$ (i.e. no observation is ever associated with the transition $q_i \rightarrow q_j$). But all of these conditions fundamentally imply $a_{ij} = 0$. Thus, a non-zero transition probability can never be updated to a zero transition probability by the B-W algorithm. Therefore, the model must be initialized with the appropriate zero transitions before training. This is equivalent to stating that the graphical structure of the HMM must be known *a priori*.